Уральский федеральный университет
имени первого Президента России Б.Н.Ельцина
Институт радиоэлектроники и информационных технологий — РТФ

О. Н. АЛЕКСАНДРОВА
С. С. ВАУЛИН
Н. В. ПАПУЛОВСКАЯ

ИНФОРМАЦИОННО-УПРАВЛЯЮЩИЕ СИСТЕМЫ: АРХИТЕКТУРА И РАЗРАБОТКА

Учебное пособие

Министерство науки и высшего образования
Российской Федерации

Уральский федеральный университет
имени первого Президента России Б. Н. Ельцина

О. Н. Александрова
С. С. Ваулин
Н. В. Папуловская

Информационно-управляющие системы: архитектура и разработка

Учебное пособие

Под общей редакцией доцента Н. В. Папуловской

Рекомендовано методическим советом
Уральского федерального университета для студентов вуза,
обучающихся по направлениям подготовки:
27.03.04 — Управление в технических системах,
09.03.03 — Прикладная информатика,
09.03.04 — Программная инженерия

Екатеринбург
Издательство Уральского университета
2021

УДК 681.518(075.8)
ББК 32.965.07я73
А46

Рецензенты:
д-р техн. наук, проф., гл. науч. сотр. Института горного дела УрО РАН *В. М. Аленичев*;
кафедра механики и автоматизации технологических процессов и производств НЧОУ ВО «Технический университет УГМК» (зав-кафедрой канд. физ.-мат. наук *П. Ю. Худяков*)

Для оформления обложки использовано изображение с сайта https://www.som.polimi.it/wp-content/uploads/2020/05/Articolo_Elia_1200x510.jpg

Александрова, О. Н.

А46 Информационно-управляющие системы: архитектура и разработка : учебное пособие / О. Н. Александрова, С. С. Ваулин, Н. В. Папуловская ; под общ. ред. доц. Н. В. Папуловской ; М-во науки и высшего образования РФ. — Екатеринбург : Изд-во Урал. ун-та, 2021. — 146, [2] с.

ISBN 978-5-7996-3338-7

Рассмотрена многоуровневая архитектура информационно-управляющих систем (ИУС), представлен обзор компонентов — аппаратуры, промышленных сетей, операционных систем реального времени, систем диспетчерского управления, систем интернета вещей. Излагается программный интерфейс для программирования систем реального времени. Рассматриваются вопросы применения облачных технологий, вопросы экономической эффективности внедрения ИУС. Изложение ведется с позиций концепции открытых систем.

Пособие предназначено для студентов всех форм обучения, а также может быть полезно специалистам в области информационных технологий.

Рис. 20. Табл. 5. Прил. 1.

УДК 681.518(075.8)
ББК 32.965.07я73

ISBN 978-5-7996-3338-7

Оглавление

Условные обозначения

Интересный факт

Знаете ли вы?

Определение

Зависимость

Это надо запомнить

Современные тенденции

Список сокращений

АРМ — автоматизированное рабочее место

АС — автоматизированная система

АСУ ТП — автоматизированные системы управления технологическими процессами

АЦП — аналого-цифровой преобразователь

АЭС — атомная электростанция

БД — база данных

Индустрия 4.0 — прогнозируемая четвертая промышленная революция с массовым внедрением киберфизических систем в производство и обслуживание человеческих потребностей

ИУС — информационно-управляющая система

ММС — магистрально-модульная система

МФИ — многофункциональный индикатор

МЭК — Международная электротехническая комиссия (МЭК, от *англ.* IEC — International Electrotechnical Comission) — некоммерческая организация, которая занимается стандартизацией всех электрических, электронных, а также смежных технологий, которые описываются в технических документах — стандартах МЭК

ОВЕН — от *англ.* Owen-cloud, облачный сервис фирмы «Овен» (Россия)

ОС — операционная система

ОС РВ — операционная система реального времени

ПДД — правила дорожного движения

ПЗУ — постоянное запоминающее устройство, энергонезависимая память с однократной записью

ПИД-регулятор — пропорционально интегрально-дифференцирующий регулятор

ПК — персональный компьютер

ПО — программное обеспечение

РАН — Российская академия наук

СУБД — система управления базами данных

ТЭС — теплоэлектростанция

УСО — устройство связи с объектом

УО — управляемый объект

ЦАП — цифроаналоговый преобразователь

ЦОД — центр обработки данных

ЭМВОС — эталонная модель взаимодействия открытых систем

ACP (Automation Collaborative Platform) — единая платформа автоматизации

ADA — язык программирования, названный в честь Ады Лавлейс, используется для программирования контроллеров и задач реального времени

AIX (Advanced Interactive eXecutive) — операционная система (семейства UNIX) производства компании IBM

AMD — компания США, производитель интегральной микросхемной электроники

API (Application Programming Interface*)* — интерфейс прикладного программирования

AS-i (Actuator-Sensor-interface) — технология промышленной сети для связи датчиков и исполнительных механизмов с контроллером

BMS (Business Management Systems) — системы управления бизнесом

C (Си) — компилируемый статически типизированный язык программирования общего назначения

C++ — компилируемый статически типизированный объектно ориентированный язык программирования общего назначения

CAMAC (Computer Automated Measurement and Control) — стандарт, определяющий организацию магистрально-модульной шины, предназначенной для связи измерительных устройств с цифровой аппаратурой обработки данных в системах сбора данных

CAN (Controller Area Network) — «сеть контроллеров», стандарт промышленной сети

CASE (Computer Aided Software/System Engeneering) — набор инструментов и методов программной инженерии для проектирования программного обеспечения

CCSLA (Cloud Computing Service Level Agreements) — стандарт, формирующий важный компонент договорных отношений между облаком

CCIP (Cloud Computing Interoperability and Portability) — стандарт, формирующий систему обмена информацией двух или более участников или компонентов и использования этой информации

CCDF (Cloud Computing Data and its Flow) — стандарт на данные облачных систем и их потоки

CDMI (Cloud Data Management Interface) — интерфейс управления облачными данными

CODESYS — инструментальный программный комплекс промышленной автоматизации, разработанный и распространяемый компанией 3S-Smart Software Solutions GmbH

COTS (commercial off the shell) — коммерчески готовые и коммерчески доступные программные продукты

CPU (central processing unit) — процессор общего назначения

CRM (Customer Relationships Management) — управление взаимоотношениями с клиентами

CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance, или Carrier sensing multiple accesses with collision avoidance) — метод управления доступом к среде передачи данных с определением несущей волны и предотвращением коллизий

DCS (Distributed Control System) — распределенная система управления

DDE (Dynamic Data Exchange) — механизм взаимодействия приложений в операционных системах Microsoft Windows и OS/2. Рассматривается как категория переменных для описания связи с внешними объектами

Decnet — семейство сетевых протоколов, разработанных компанией Digital Equipment Corporation и первоначально выпущенных в 1975 г.

DPC (Deferred procedure call) — специфический механизм вызова процедур в архитектуре Windows

ERP (Enterprise Resource Planning) — организационная стратегия интеграции производства, управления трудовыми ресурсами, финансового менеджмента и управления активами, ориентированная на непрерывную балансировку и оптимизацию ресурсов предприятия

FASTBUS — стандарт локальной магистрали и крейтовой системы, ориентированный на работу в высокоскоростных системах сбора данных

FBD (Function Block Diagrams) — язык функциональных блоков стандарта IEC-61131

FIFO (First In First Out) — дисциплина диспетчеризации, при которой процесс из очереди готовых к выполнению процессов данного приоритета получает управление и выполняется до тех пор, пока не завершится

Fieldbus — открытая архитектура цифровой, последовательной, двусторонней системой связи, которая служит в качестве базового уровня сети в промышленных системах автоматизации

FPGA (Field-programmable gate array) — специализированные программируемые логические матрицы

HMI (Human-machine interface) — человеко-машинный интерфейс

HTTP (HyperText Transfer Protocol) — протокол прикладного уровня передачи данных

GPU (Graphics processing unit) — графический процессор

GSM (Global System for Mobile Communications) — глобальный стандарт цифровой мобильной сотовой связи

IaaS (Infrastructure-as-a-Service) — модель использования облачных технологий, при которой предоставляется возможность вынести IT-инфраструктуру предприятия на специально подготовленные облачные виртуальные серверы

IBM (International Business Machines) — американская компания, производитель аппаратного и программного обеспечения, IT-сервисов и консалтинговых услуг

ICS (Industrial Control System) — промышленные системы безопасности

IETF (Internet Engineering Task Force) — открытое международное сообщество проектировщиков, ученых, сетевых операторов и провайдеров, занимающееся развитием протоколов и архитектуры интернета

IL (Instruction List) — язык инструкций стандарта IEC-61131

InterBUS — технология промышленной сети

IoT (Internet of Things) — интернет вещей

IBoT (Internet of Battle Things) — интернет вещей в военной промышленности

IIoT (Industrial Internet of Things) — интернет вещей в промышленности

IsaGraf/IsaGraph — инструментальная графическая среда разработки прикладных программ для программируемых логических контроллеров

ISR (Interrupt Service Routine) — обработчик прерывания

LD (Ladder Diagrams) — язык релейных диаграмм (релейной логики) стандарта IEC-61131

MES (Manufacturing Execution System) — система управления производственными процессами

MIPS (Microprocessor without Interlocked Pipeline Stages) — система команд и микропроцессорных архитектур, разработанных компанией MIPS Computer Systems в соответствии с концепцией проектирования процессоров RISC

MOTS (Military off-the-shelf) — аппаратура либо программные продукты, предназначенные для военного использования

MRP (Manufacturing Resource Planning) — система планирования потребностей в материалах, одна из наиболее популярных в мире логистических концепций

NB-IoT (Narrow Band Internet of Things) **—** стандарт сотовой связи для устройств телеметрии с низкими объемами обмена данными. Разработан консорциумом 3GPP в рамках работ над стандартами сотовых сетей нового поколения

NDI (Non-Developmental Item) — принцип неразрабатываемых заново технологий

NetDDE — служба сетевого динамического обмена данными, обеспечивает сетевой транспорт и безопасность для программ, выполняющихся на одном или на различных компьютерах

NIM (Nuclear Instrumentation Module) — стандарт, задающий механическую и электрическую спецификацию для модулей-крейтов, используемых в оборудовании для обеспечения экспериментов физики элементарных частиц и ядерной физики

ODBC (Open Database Connectivity) — программный интерфейс (API) доступа к базам данных, разработанный компанией Microsoft в сотрудничестве с Simba Technologies на основе спецификаций Call Level Interface (CLI)

OLE (Object Linking and Embedding) — технология связывания и внедрения объектов в другие документы и объекты, разработанная корпорацией Microsoft

OMI (Operation Managemant Interface) — интерфейс управления операциями в промышленности

OCP (Open Compute Project) — концепция, сообщество и организация, в рамках которых участники в форме открытого диалога делятся разработками в сфере программного, аппаратного и физического проектировании современных дата-центров

OPC (Open Platform Communications) — протокол, предоставляющий единый интерфейс для управления объектами автоматизации и технологическими процессами

OPC-сервер — технологии OPC для связи контроллеров со SCADA

OPC UA (OPC unified atchitecture) — технология универсального интерфейса для взаимодействия аппаратного и верхнего уровней автоматизации предприятий

OSI (Open Systems Interconnection) — модель сетевого взаимодействия открытых систем, определяет функции взаимодействия данных на разных уровнях

OutDoor Box — промышленный микрокомпьютер IP66 для интерфейса нейросетей при обработке 4 потоков

OVF (Open Virtualization Format) — открытый стандарт для хранения и распространения виртуальных машин

PaaS (Platform as a Service) — модель использования облачных технологий, при которой потребитель получает у облачного провайдера готовую платформу; может управлять данными, устанавливать и конфигурировать свои приложения

PCI (Peripheral component interconnect) — шина ввода-вывода для подключения периферийных устройств к материнской плате компьютера

PDP (Programmed Data Processor) — серия мини-компьютеров, разработанных компанией Digital Equipment Corporation

PDO (Process Data Objects) — механизм, используемый для передачи с высокой скоростью высокоприоритетных данных

P-NET — технология промышленной сети

POSIX (Portable Operating System Interface) — переносимый интерфейс операционных систем (набор стандартов)

Profibus FMS (Process Field Bus — шина полевого уровня и Fieldbus Message Specification — спецификация сообщения полевой шины) — протокол передачи данных, предназначенный для связи программируемых контроллеров друг с другом и со станциями оператора

Profibus DP (Process Field Bus и Decentralized Peripherals) — профиль протоколов промышленной сети для взаимодействия периферийного оборудования на полевом уровне

Profibus PA (Process Automation) — промышленная сеть, которая служит для соединения систем автоматизации и систем управления процессами с полевыми устройствами (сенсорами и др.) во взрывопожароопасных производствах

PROFINET — технология промышленной сети

QNX — POSIX-совместимая операционная система реального времени

QR-код (Quick Response Code — код быстрого реагирования) — тип матричных штрихкодов

REST (Representational State Transfer) — архитектурный стиль взаимодействия компонентов распределенного приложения в сети

RISC (Reduced Instruction Set Computer) — архитектура процессора, в которой быстродействие увеличивается за счет упрощения инструкций

ROTS (Rugged Off-The-Shelf) — аппаратура или программные продукты повышенной надежности для работы в сложных атмосферных условиях

RPA (Robotic Process Automation) — роботизация бизнес-процессов

RTU (Remote Terminal Unit) — устройство связи с объектом (УСО, модуль ввода-вывода)

RS-485 (Recommended Standard 485) — стандарт физического уровня для помехоустойчивого асинхронного интерфейса

SaaS (Software as a Service) — модель использования облачных технологий, при которой потребителю предоставляется готовое прикладное программное обеспечение, полностью обслуживаемое облачным провайдером

SANS — Институт SANS (Институт передовых технологий) — частная американская коммерческая компания, специализируется на образовании в области информационной безопасности

SCADA (Supervisory Control And Data Acquisition) — класс систем диспетчерского управления, сбора, обработки, отображения и архивирования информации об объекте

SCI (Scalable Coherent Interface) — специализированный стандарт вычислительной сети в 1990-х гг.

SCOR (Supply-Chain Operations Reference) — модель процессов цепочки поставок

SCM (Supply Chain Management) — система управления процессами поставок товара, организованная сеть сбыта

SCTP (Stream Control Transmission Protocol) — протокол транспортного уровня в компьютерных сетях, разработанный IETF в 2000 г. для использования в системах связи

SDO (Service Data Objects) — механизм, используемый для конфигурирования устройств низкой приоритетности

SFC (Sequential Function Charts) — язык последовательных функциональных схем стандарта IEC-61131

SIGRTMIN и SIGRTMAX — первое и последнее значение диапазона номеров пользовательских сигналов реального времени (символические константы), используемых в POSIX-совмесимых ОС

SOFTLOGIC — российская IT-компания, осуществляющая разработку и внедрение программно-аппаратных комплексов с использованием искусственного интеллекта, проектирование и поставку высокотехнологичного оборудования для бизнеса и государственного сектора

SPARC (Scalable Processor Arhitecture) — архитектура RISC-микропроцессоров, первоначально разработанная в 1985 г. компанией Sun Microsystems

SQL (Structured Query Language) — язык структурированных запросов к СУБД

ST (Structured Text) — язык структурированного текста стандарта IEC-61131

TIC (Target Independent Code) — платформенно независимый код систем программирования стандарта IEC-61131

TCP/IP (Transmission Control Protocol (TCP) и Internet Protocol (IP)) — стек протоколов интернета

Trace Mode — программный комплекс класса SCADA HMI, разработанный компанией AdAstra Research Group

UML (Unified Modeling Language) — унифицированный язык моделирования

UNIX — зарегистрированная торговая марка организации The Open Group — семейство переносимых, многозадачных и многопользовательских операционных систем

VAX (Virtual Address eXtension) — 32-битная компьютерная архитектура, разработанная в 1970-х гг. компанией Digital Equipment Corporation и являющаяся развитием линии PDP-11

VME (Versa Module Eurocard bus) — стандарт на компьютерную шину, первоначально разработанный для семейства микропроцессоров Motorola 68000

VMS (VME Serial) — высокоскоростная последовательная подсистема шины VME, используемая в качестве промежуточных соединений ввода-вывода и для передачи данных

VPX (также известный как VITA 46) — стандарт ANSI (ANSI/VITA 46.0–2019), обеспечивающий системы на основе VMEbus поддержкой коммутируемых структур или фабрик через новый высокоскоростной соединитель.

VXI (VMEbus eXtention for Instrumentation) — одно из направлений развития шины VMEbus

WRT (WebSphere Real-Time Java) — технология, позволяющая писать программы реального времени, содержит автономную среду выполнения Java (JRE) на базе Java Standard Edition v5

Предисловие

Для успешного функционирования предприятий в условиях жесткой конкуренции, жизненно важным является применение систем, автоматизирующих технологические и бизнес-процессы. Знания в этой области являются фундаментальными для IT-специалистов, разработчиков прикладных информационно-управляющих систем. Данное учебное пособие написано на основе лекционного материала по дисциплинам «Информационно-управляющие системы», «Автоматизированные системы управления технологическими процессами», «Автоматизация технологических процессов», «Системы управления предприятием».

Цель пособия — дать представление о современном уровне автоматизации на промышленных предприятиях, сформировать компетенции в области построения и разработки информационно-управляющих систем на разных уровнях управления данными и оборудованием.

Задачи пособия:

- познакомить читателя с назначением, архитектурой, особенностями реализации информационно-управляющих систем (ИУС);
- изложить особенности построения и разработки ИУС с применением концепции открытых систем;
- восполнить имеющийся пробел в доступной литературе, рассматривающей указанные вопросы с системных позиций.

Учебное пособие состоит из двух разделов. Раздел А содержит теоретические сведения, необходимые для понимания принципов и подходов к построению информационно-управляющих систем. Раздел Б имеет практическое значение. В нем описаны примеры использова-

ния и настройки системы реального времени и представлены рекомендации к разработке информационно-управляющих систем. В конце каждой главы размещены контрольные вопросы для самостоятельной проверки усвоения теоретического материала, в конце раздела Б читателям предлагается практическое задание.

Завершает пособие список организаций, комитетов и комиссий, формирующих стандарты в области информационных систем и технологий.

Авторы

Введение

При разработке современных автоматизированных систем (АС) целесообразно применять комплексный подход, предусматривающий решение задач автоматизации на всех уровнях системы — от финансового планирования ресурсов предприятия до непосредственного управления технологическим оборудованием. Такие системы традиционно называются автоматизированными системами управления технологическими процессами (АСУ ТП). Сейчас чаще используется термин «информационно-управляющие системы» (ИУС), поскольку они — ИУС — применяются не только в промышленном производстве, но и в других сферах деятельности человека, в т. ч. в быту, спорте, на транспорте и т. д. Автоматизация коснулась многих сфер жизни и деятельности человека. Если раньше автоматизированные системы управления могли иметь только крупные производства, то сегодня без таких систем невозможно представить любое производство. Таким образом, **информационно-управляющие системы** — это системы автоматизации, применяемые в самых различных отраслях: в энергетике и добывающей промышленности, отраслях водного транспорта, авиации, космонавтике, жилищно-коммунальном хозяйстве, животноводстве, в отрасли биржевых операций и мн. др. Для наглядности приведем примеры некоторых ИУС.

Классическим примером ИУС (обычно учебники и пособия начинаются с описания именно таких систем) является система управления ректификационной колонной, которая разделяет нефть на тяжелые и легкие фракции при ее перегонке. Система, основываясь на текущих значениях скорости, температуры и давления парожидкостной

смеси нефти, претерпевающей ректификацию, выдает команды на регулировку параметров [1]. В результате данного процесса изменяются нагрев исходного сырья, положение задвижек трубопроводов, соединяющих различные секции колонны, степень рециркуляции паров отдельных фракций и т. д. ИУС ректификационной колонны максимизирует выход готового продукта (объемы выделяемых фракций) и снижает расход продуктов на нагрев сырья.

Следующий пример ИУС — система управления движением поездов. Сегодня мы с трудом представляем, что стрелки на железнодорожных путях переводит человек. Это, во-первых, очень опасно, а во-вторых, неэффективно. Протяженность железнодорожной сети общего пользования в России превышает 85 тыс. км. Это огромная транспортная сеть, на которой ежедневно перемещаются тысячи поездов.

Типичным примером ИУС является система управления энергоблоками. Использование ИУС позволяет оптимизировать процесс выработки электроэнергии и тепла при минимизации расхода топлива (угля, мазута, газа).

В бытовой технике элементы ИУС применяются очень широко. Типичный пример — робот-пылесос, для которого можно задать программу уборки помещения, и он ее выполнит в заданное время.

В спортивной технике размещаемые на спортсмене датчики и контроллер с радиосвязью позволяет тренеру дистанционно контролировать состояние и действия спортсмена. Так, в парусном спорте элементы ИУС применяются уже десятки лет. Типичный пример — оборудование яхт для одиночных кругосветных и трансокеанских гонок.

Еще одним типичным примером ИУС является электродистанционная система управления самолетами.

Современные самолеты, как правило, проектируются как аэродинамически неустойчивые системы, а их устойчивость в полете обеспечивается системой управления. Применение ИУС на широкофюзеляжных пассажирских самолетах позволяет снижать размеры стабилизирующих плоскостей и, следовательно, экономить топливо, а у боевых самолетов удается повышать их маневренность.

Исключительно ручное пилотирование современными лайнерами, когда ИУС выведена из строя, например, ударом молнии или по какой-либо другой причине, является сложным для рядового летчика и может закончиться катастрофой.

Именно так над Атлантикой в 2009 г., после удара молнии, потерпел крушение французский авиалайнер А-330. Имело место отключение бортовой ИУС. Пилоты не смогли справиться с ручным управлением воздушным судном, и авиалайнер разбился о поверхность воды. По аналогичной причине (пилоты не справились с ручным управлением самолетом при отказе системы управления) произошла катастрофа SSJ-100 в аэропорту Шереметьево (2019).

Система управления самолетом (бортовая ИУС) делает его устойчивым. Впервые такие стабилизирующие системы, использующие разрезные тяги управления, были применены на тяжелых вертолетах Як-24. В современных так называемых электродистанционных системах управления полностью отсутствует механическая связь органов управления в кабине экипажа с аэродинамическими поверхностями управления. У пилота имеется своеобразный «джойстик», вырабатывающий воздействие на систему управления (ИУС).

У американского истребителя четвертого поколения F-16 такой «джойстик» перенесен на правый подлокотник кресла пилота. Это позволяет передавать существенное воздействие на управляющие плоскости при небольшом воздействии, задаваемом пилотом. Пилотажные характеристики современных боевых самолетов в значительной мере определяются характеристиками системы управления. Кроме того, ИУС обеспечивает для пилота адекватное отображение информации о состоянии летательного аппарата и пространства вокруг него.

На самолетах (впервые на МИГ-31) и космических аппаратах используется и речевое оповещение о событиях на летательном аппарате и вокруг него. Примером тому является бортовая ИУС российских спутников, исследовавших в 1980-х гг. самый удаленный радиационный пояс Земли (его высота составляет порядка 17 000 км). Информация со спутника поступала в центр управления в виде голосовых сообщений.

Современные ИУС по управлению учебно-боевыми самолетами, например бортовая ИУС Як-130 и его аналогов, могут имитировать особенности пилотирования самолетов самых различных типов, меняя динамические характеристики алгоритма управления.

В качестве объекта управления могут быть и бизнес-процессы, в таком случае цель системы состоит в осуществлении автоматизированного планирования и контроля производственной деятельности пред-

приятия. Примером этому может служить система ERP. Такая ИУС с определенной периодичностью осуществляет сбор, обработку и предоставление информации о производственно-технологических процессах, а также предоставляет возможность планирования и управления функциональными процессами организации.

В настоящем пособии рассматриваются характеристики прежде всего полноценных индустриальных ИУС, работающих в составе комплексной АС предприятия.

Отметим предпосылки для внедрения подобных ИУС:

- техническая возможность и целесообразность автоматизации технологического процесса;
- экономическая целесообразность автоматизации технологического процесса;
- требования к обеспечению безопасности;
- экологические требования.

Раздел А
Архитектура информационно-управляющих систем

Информационно-управляющая система — это программно-аппаратный комплекс, центральным элементом которого является вычислительный блок. В зависимости от решаемой задачи, вычислительный блок может быть либо простейшей платой с микроконтроллером, либо многопроцессорным комплексом с внешней памятью большого объема. Вычислительный блок решает две задачи:

- реализует автоматическое программное управление в режиме реального времени;
- организует взаимодействие (интерфейс) аппаратно-программного комплекса с обслуживающим персоналом. В рамках этой задачи визуализируется состояние объекта управления путем вывода его параметров и статистических данных на монитор диспетчера и предоставляются средства ручного управления объектом.

Кроме вычислительного блока, ИУС имеет блок генерации данных, блок вывода данных, блок воздействия на объект управления, базу данных, систему информационно-телекоммуникационной инфраструктуры — инфокоммуникаций, силовые системы. Количество и состав блоков в ИУС зависит от объекта управления и назначения системы. Далее подробно разберем функциональные особенности ИУС и архитектурные подходы разработки.

1.1. Особенности функционирования информационно-управляющей системы

При создании ИУС используются те же принципы, что при создании любой автоматизированной системы (АС). Однако есть целый ряд особенностей, которые определяются тем, что работа ИУС связана с необходимостью обеспечить управление объектом в режиме реального времени. Управляемым объектом (УО) может быть все, что нас окружает: энергоблок электростанции, летательный аппарат, прокатный стан, система городских светофоров, животное, бизнес-процессы

и т. д. Приведем несколько принципиальных характеристик ИУС, которые требуют особого рассмотрения на этапе проектирования:

- работа промышленных ИУС, в отличие от офисных компьютеров, осуществляется в тяжелых (агрессивных) для электронной техники условиях внешней среды, поэтому они должны иметь повышенную термо-, вибро-, ударопрочность, пыле- и влагозащищенность;
- в масштабных системах количество каналов ввода-вывода может составлять десятки и сотни тысяч. В связи с этим требуется подключать широкую номенклатуру внешних устройств, использовать принцип модульного построения аппаратуры и выполнять взаимодействие (сопряжение) программного обеспечения с помощью большого количества драйверов;
- время реакции ИУС на изменения параметров объекта управления определяется короткими временны́ми интервалами. У особо ответственных систем, например при управлении летательным аппаратом, реакция должна быть практически мгновенной. Это предполагает повышенную надежность и аппаратной, и программной частей ИУС. Наработка на отказ на уровне модуля обычно составляет, как правило, не менее 100 000 ч, что соответствует примерно 10 годам непрерывной работы. Некоторые производители декларируют время наработки на отказ до миллиона часов.

Работа в реальном времени не означает темп работы (быстрая или медленная работа), т. к. временно́й масштаб события на управляемом объекте может варьироваться в широком диапазоне: микросекунды для летательного аппарата и десятки секунд для управления аудиоинформацией. Например, ИУС должна обработать аудиоинформацию длительностью 10 с некоторого звукового спектра. Если для анализа и обработки ей потребуется 10,01 с, то эта обработка уже не является процессом реального времени. Если же ей для обработки потребуется 9,99 с, то мы получим процесс реального времени. Говоря о реальном времени, мы подразумеваем количественную характеристику, которая может быть измерена реальными физическими часами. Существуют международные и национальные стандарты, определяющие реальное время. Стандарт POSIX 1003.1 определяет **реальное время** в операционных системах как «способность операционной системы обеспечить требуемый уровень сервиса в заданный промежуток времени». Для ра-

боты ИУС в реальном времени приходится решать проблемы привязки внутрисистемных событий к реальным моментам времени.

Работа в реальном масштабе времени оценивается с помощью следующих параметров: дедлайна (deadline), латентности (latency), джиттера (jitter) и др. Под **дедлайном** подразумевается критический временной интервал обслуживания или предельный срок завершения какой-либо работы. Этот временной интервал определяется характерным временем развития события в объекте или технологическом процессе $\tau_{\text{событие объекта}}$, которое может привести к разрушению объекта или процесса. **Латентность** указывает время отклика системы на событие в целом $\tau_{\text{ИУС}}$. **Джиттер** характеризует разброс времен отклика системы, который может возникнуть, например, под влиянием других событий, происходящих в объекте или технологическом процессе.

В зависимости от допустимости нарушений временных ограничений и цены этих нарушений в финансовом и общечеловеческом смысле, выделяют системы жесткого реального времени и системы мягкого реального времени.

Система жесткого реального времени должна без сбоев отвечать на внешние события в рамках заранее определенного интервала времени. Время ответа должно быть предсказуемым и не зависеть от текущего состояния системы. Мягкая ОС РВ тоже должна отвечать очень быстро, но гарантированное время ответа в ней не обеспечивается.

В жесткой системе:

- опоздания не допускаются ни при каких обстоятельствах;
- в случае опоздания, результаты обработки уже никому не нужны;
- опоздание считается катастрофическим сбоем;
- стоимость опоздания бесконечно велика.

Типичным примером жесткой системы реального времени может служить система управления движением воздушных судов. Очевидно, что бессмысленно посылать команду на изменение курса самолета после того, как он потерпел крушение в результате столкновения.

В системе мягкого реального времени:

- опоздание не считается катастрофическим сбоем;
- вводится высокая, но конечная стоимость опоздания;
- допускается низкая производительность в случае опоздания.

Системы мягкого реального времени могут использоваться, например, в телекоммуникационных системах. Маловероятная задержка небольшой части пакетов обычно вполне допустима и не приводит

к катастрофическим последствиям. Отказ от обязательного отсутствия опозданий упрощает архитектуру системы и делает ее экономичнее в части потребления ресурсов.

1.2. Многоуровневый иерархический подход при построении ИУС

Традиционно при построении ИУС используют многоуровневый иерархический подход. В системах промышленной автоматизации можно выделить пять логических уровней (рис. 1). ИУС включает следующие уровни (снизу вверх):

I — уровень ввода-вывода. Уровень устройств, непосредственно взаимодействующих с управляемым объектом, датчиков и исполнительных механизмов (актуаторов);

II — уровень управления вводом-выводом. На нем представлены контроллеры предварительной обработки информации различной сложности;

III — уровень сбора и диспетчерского управления данными (SCADA — Supervisory Control And Data Acquisition);

IV — интеллектуальный уровень аналитики данных, или уровень оперативного управления производством (MES — Manufacturing Execution System);

V — уровень планирование ресурсов предприятия (ERP — Enterprise Resource Planning).

Далее рассмотрим особенности каждого уровня.

Информация об объекте собирается датчиками (*уровень ввода-вывода*). Между датчиками и исполнительными устройствами, с одной стороны, и устройствами цифровой обработки — с другой, устанавливаются аналого-цифровые (АЦП) и цифроаналоговые (ЦАП) преобразователи. Эти операции могут выполняться также специализированными контроллерами.

Датчики могут быть двух типов: пассивные и активные.

В случае использования пассивного датчика, система сама периодически считывает с него информацию. К таким датчикам относятся, например, датчики, измеряющие температуру, влажность, освещенность в некоторой точке управляемого объекта.

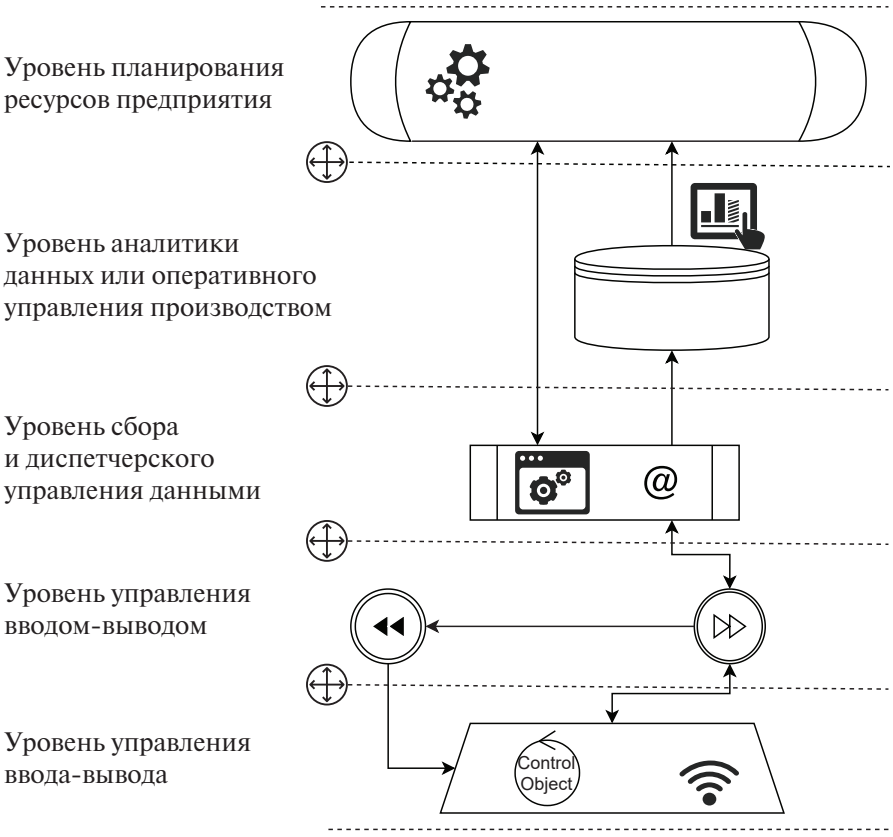

Рис. 1. Модель информационно-управляющей системы

Примером активного датчика является датчик цепной реакции в хранилище ядерного топлива. При изменении своего состояния, активный датчик самостоятельно сообщает об этом контроллеру, используя механизм прерываний. После получения сообщения активного датчика об аварийной ситуации, система вырабатывает воздействие на регулируемый процесс, которое осуществляется с помощью электрических или электромеханических исполнительных механизмов. Например, это может быть введение дополнительных стержней (материалов) поглощения нейтронов или включение-выключение устройства для регулирования температуры.

На уровне *управления вводом-выводом* решаются следующие задачи:

- прием сигналов с датчиков, аналого-цифровое преобразование сигналов, нормализация и другая обработка полученной информации;

- передача (уже цифровой) информации о состоянии управляемого объекта на вышестоящие уровни системы;
- прием информации о воздействии на управляемый объект с более высоких уровней системы, преобразование информации в аналоговую форму, передача сформированных сигналов на исполнительные механизмы;
- возможно локальное автоматическое управление некоторыми параметрами управляемого объекта.

Получая информацию от датчиков, контроллеры формируют значения состояния управляемого объекта. Часть этих значений соответствует измеряемым параметрам. В таком случае от контроллера требуется нормализация и приведение данных к необходимому формату для передачи. Некоторые переменные состояния объекта не измеряются. Их значения вычисляются на основе других данным. Примером такого параметра является уровень воды в барабане парового котла. Этот параметр является ключевым для алгоритма автоматического управления, но его не измеряют из-за высоких значений температуры (более 500 °С) и давления (более 100 атм), а вычисляют, используя данные о давлении и температуре пароводяной смеси в различных точках барабана.

Являясь специализированными компьютерами, контроллеры ИУС отличаются от ЭВМ общего назначения условиями работы и требованиями к выполнению задач:

- контроллеры обычно работают в тяжелых (для электронной техники) условиях окружающей среды с широким диапазоном температур, высокой влажностью, запыленностью и при высоком уровне электромагнитных помех;
- контроллеры должны обслуживать, как правило, большое количество каналов ввода-вывода, что требует модульного исполнения аппаратуры и наличия соответствующих драйверов в программном обеспечении;
- контроллеры имеют небольшие вычислительные ресурсы и реализуют относительно простые алгоритмы работы, не требующие большой вычислительной мощности;
- контроллеры обязаны обеспечить обмен информации с датчиками и исполнительными механизмами в темпе, который определяется динамикой управляемого объекта.

В системе, как правило, устанавливается избыточно большое количество контроллеров. Это позволяет распараллелить

операции обработки информации, снизить требования к ресурсам контроллеров, увеличить их надежность вследствие устранения единых точек отказа и уменьшить длину аналоговых линий связи от датчиков до контроллеров. Последний фактор имеет важное значение. Например мы измеряем термопарой температуру в некоторой точке турбогенератора. Термопара генерирует ЭДС в несколько десятков милливольт. Рядом вращается ротор с сильноточными обмотками, создающий сильнейшие переменные электромагнитные поля. В таком случае линию передачи сигнала от термопары следует защищать от помех, создаваемых электромагнитным полем ротора, что является весьма сложной задачей.

Иногда все контроллеры перемещают в чистое помещение, так что длина линий связи составляет несколько сот метров, а компьютеры помещают в кожухи, в которые подается специально очищенный воздух. Именно таким способом разработчики защитили аппаратный комплекс ИУС на Воскресенском заводе минеральных удобрений, в производственных цехах которого имела место крайне агрессивная атмосфера.

Контроллер должен быть готовым не только к работе в агрессивной среде и сложных условиях, но и иметь развитую систему прерываний и обеспечивать работу в реальном времени. Для выполнения задач в реальном масштабе времени, на контроллере устанавливают операционную систему реального времени (ОС РВ), которая может быть как полноценно развитой многозадачной ОС РВ, так и простейшим управляющим (runtime) модулем, который часто компонуют вместе с прикладным кодом. В ОС РВ QNX шестой версии такой модуль называется System Process. В качестве ОС РВ достаточно часто пытаются использовать соответствующим образом сконфигурированные ОС общего назначения, что допустимо при невысокой динамике управляемого объекта и наличии в них средств поддержки реального времени. Однако в этом случае возможна реализация только системы мягкого реального времени. Отметим, что, как правило, для разработки прикладного программного обеспечения (ПО) контроллера приходится применять специализированные средства и среды разработки.

Для передачи информации от контроллеров на вышестоящие уровни системы (диспетчерский уровень и др.), их соединяют вычислительной сетью и используют специализированные технологии —

технологии промышленных сетей (*англ.* Fieldbus, что означает сети, способные работать в полевых условиях). Речь идет о технологиях PROFIBUS/PROFINET, P-NET, INTERBUS, CAN и т. д.

Применение типовых промышленных сетей при построении систем общего назначения обычно является нецелесообразным в связи со сложностью и ресурсоемкостью их технологий. Укажем основные отличительные особенности технологии промышленных сетей:

- промышленные сети обеспечивают работу в тяжелых условиях окружающей среды;
- обеспечивают предсказуемость времени передачи данных. Обычно при использовании разделяемой среды передачи данных применяется либо маркерный метод доступа, либо технология CSMA/CA. Ethernet может использоваться только в коммутируемом варианте;
- операции по передаче данных существенно отличаются от операций интерфейса сокетов, т. к. сети являются узкоспециализированными. В качестве базовых операций используются чтение размещенной на объекте переменной состояния (т. е. передача информации от контроллера к центральному узлу) и запись переменной состояния (т. е. передача информации от центрального узла к контроллеру). Применяются относительно короткие кадры передачи информации с простой структурой, поскольку объем передаваемой за одну операцию данных небольшой;
- структура стеков протоколов промышленных сетей существенно отличается от сетей общего назначения. Отсутствуют, как правило, средние уровни. Не требуются такие операции, как маршрутизация. В то же время весьма развит прикладной уровень. Именно на прикладном, а не на транспортном уровне обеспечивается надежная передача данных и определяются профили устройств, которые включают типовые наборы переменных состояния для стандартизированных классов устройств;
- для большинства технологий допускается применение упрощенных узлов сети (пассивных узлов, Slave-узлов), реализующих, в отличие от полноценных узлов (активных узлов, Master-узлов), часть сетевого стека протоколов. При маркерном способе доступа, маркер передается только между активными узлами. Получив доступ к сети, Master-узел опрашивает подчиненные ему Slave-узлы;

- скорости передачи данных в промышленных сетях обычно невысокие из-за малого объема передаваемых за одну операцию данных. Увеличения пропускной способности не требуется.

В качестве примера упрощенных технологий промышленных сетей приведем интерфейс ASi с одним активным устройством и простыми форматами кадров. Такой интерфейс используется для передачи информации либо между контроллером и датчиком или исполнительным механизмом, либо между контроллером и групповым узлом, к которому датчики и исполнительные механизмы подключаются с помощью коротких аналоговых линий.

Технологии промышленных сетей описываются стандартом IEC 61158. Стандарт является многофункциональным, т.е. описывает сразу несколько технологий, включая стандарт на телекоммуникационные профили IEC 61784.

Таким образом, на уровне управления вводом-выводом используются:

- специализированная аппаратура;
- специализированные сетевые технологии;
- специализированное системное программное обеспечение (ОС РВ);
- специализированное прикладное программное обеспечение (инструментальные системы для программирования контроллеров).

На *уровне сбора и диспетчерского управления* (см. рис. 1) данными решаются следующие задачи:

- сбор и обработка данных от контроллеров;
- отображение результатов обработки данных для персонала;
- обработка действий персонала по управлению технологическим объектом;
- управление объектом и передача соответствующих команд контроллерам;
- накопление результатов обработки данных, действий персонала и системы.

На этом уровне реализуются алгоритмы управления объектом, замыкаются главные контуры автоматического управления и обязательно предусматриваются средства ручного (на данном уровне предполагается присутствие хотя бы одного человека-оператора-диспетчера) или полуавтоматического (система корректирует действия оператора) управления объектом.

На уровне сбора данных возможно использовать аппаратуру и системное программное обеспечение различных типов. Это зависит от типа управляемого объекта и его динамики, места размещения серверов, рабочих станций и промежуточных устройств.

Если мы, например, управляем подвижным объектом с высокой динамикой (беспилотным летательным аппаратом), а средства диспетчерского управления находятся на самом объекте или на наземном подвижном пункте управления, то для построения ИУС требуется специализированная аппаратура и специализированное системное ПО (промышленные компьютеры и ОС РВ). Если же мы управляем подвижным объектом с высокой динамикой, используя средства диспетчерского управления наземного стационарного пункта, то в таком случае также требуется специализированная ОС РВ, но аппаратура может быть и обычной — аппаратурой общего назначения, т. к. условия окружающей среды это позволяют.

В то же время требования реального времени, которые обеспечиваются и аппаратной составляющей ИУС, безусловно, остаются. Например, при решении некоторой прикладной задачи возникают аппаратные неисправности, которые не могут быть устранены за время, определяемое динамикой объекта (а это очень небольшие интервалы времени). Как результат, мы теряем аппарат, не выполняем задачу и, возможно, наносим ущерб другим объектам. Для исключения таких ситуаций, к аппаратуре общего назначения применяют средства, позволяющие реализовать так называемую систему высокой готовности (High Availability), главным параметром которой является коэффициент готовности $K_{\text{Г}}$. Он определяется по отношению времени простоя $T_{\text{пр}}$ из-за неисправности к общему астрономическому времени работы системы $T_{\text{общ}}$

$$K_{\text{г}} = 1 - T_{\text{пр}} / T_{\text{общ}}.$$

Например, при коэффициенте готовности «пять девяток» (0.99999) время простоя составляет примерно 5 минут в год. Это достигается путем горячего резервирования основных компонентов системы или системы целиком, т. е. использованием для восстановления работоспособности системы перехода на использование резервных компонентов с последующей заменой отказавшего компонента без ее выключения. Эти механизмы требуется реализовать как на уровне прикладных, так и на уровне системных программ.

Если удается реализовать систему так, чтобы время переключения на резервную систему у аппаратуры общего назначения было бы меньше допустимого времени задержки, то такая аппаратура может быть применима. Отметим, что использование специализированной промышленной аппаратуры не исключает необходимости применения средств обеспечения высокой готовности. На диспетчерском уровне используется достаточно много аппаратуры, включая порой мощные многопроцессорные серверы и большое количество станций оперативного персонала. В результате этого даже при приемлемой наработке на отказ на уровне модуля, типовые значения которой составляют 100–300 тыс. ч безотказной работы, коэффициент готовности всей системы без резервирования может получиться неприемлемым.

В случае управления стационарным объектом, изменение состояния которого отличается невысокой динамикой (например водогрейным котлом), а средства диспетчерского управления находятся в стационарном отапливаемом помещении, для построения ИУС мы можем использовать аппаратуру и системное ПО общего назначения. Отметим, что многие инструментальные системы диспетчерского управления работают под управлением ОС Windows.

Разработка прикладного программного обеспечения данного уровня может быть выполнена с использованием любого универсального инструментария. Однако часто это приводит к неприемлемым срокам разработки, поэтому практически всегда используются специализированные инструментальные системы. Типичными представителями таких систем являются InTouch (Wonderware, США) и TraceMode (AdAstra Research Group, Россия). Обе системы работают под управлением MS Windows. Исходно обе системы представляли собой средство создания интерфейса оператора, но затем постепенно переросли в системообразующий компонент, который, кроме того, может работать на смежных уровнях.

Применение инструментальных систем SCADA позволяет за 2–3 мес. разработать и внедрить типовую ИУС, начиная от утверждения технического задания и заканчивая пуском системы в эксплуатацию и обучением персонала. Это многократно подтверждалось на примере разных проектов.

При создании графического пользовательского интерфейса определяются переменные состояния управляемого объекта. В терминологии SCADA они называются переменными управления. Состояния пере-

менной могут быть связаны либо с измеряемым параметром (значение переменной получаем от контроллера), либо с устанавливаемым параметром (значение переменной пересылается на контроллер), либо вычисляться в программе [1]. Переменные управления типизированы и имеют целый набор атрибутов: наличие предупредительного сообщения, вызванного выходом значения за границы установок, признак квитированности этого сообщения, групповая принадлежность переменной, комментарий и т. д.

При создании интерфейса, переменной ставится в соответствие графический образ, который размещается в рабочем окне и визуализирует значение переменной и ее атрибуты. В качестве атрибута может быть предупредительное сообщение. Графический образ может быть составным. В этом случае в него входят элементы для изменения значений переменной. Например, графический образ может соответствовать диалоговым панелям Windows. Между переменной и графическим образом устанавливаются анимационные связи, изменяющие внешний вид образа в зависимости от значения переменной. На экране это отображается как изменение размера, цвета, положения, мигание, вращение образа. В составе любой системы SCADA поставляется расширяемая библиотека графических образов, включающая панели, лампочки, тренды, измерительные линейки, часы, переключатели, клавиши и т. д. Визуализация данных реализуется либо в числовой форме, либо в графической, либо в виде графика (тренда), изменяющегося в реальном времени.

Все вводимые разработчиком переменные заносятся в базу данных, которая в целевой системе начинает работать как база данных реального времени. Обычно она является специализированной и входит в состав инструментального пакета. Система управления базами данных реального времени отличается от обычной СУБД тем, что все записи БД обязательно содержат метки времени и качества, т. к. для задач реального времени важно не только значение того или иного параметра, помещаемое в БД, но и момент записи, т. е. момент времени измерения этого параметра. Примером базы данных реального времени является СУБД реального времени Industrial SQL Server (от компании Wonderware). IndustrialSQL Server представляет собой реляционную БД, в которой учтена скорость поступления и объемы производственной информации. IndustrialSQL Server, интегрированный со SCADA-компонентом InTouch, способен накапливать при помощи серверов

ввода-вывода информацию практически от любых измерительных приборов и устройств сбора данных.

Разработка графического образа переменной управления входит в задачи системы InTouch. Переменные могут принадлежать к одной из двух категорий: категории DDE (для описания связи с внешними объектами) либо категории Memory (для внутренних переменных). Кроме значения, переменная имеет набор атрибутов. Переменную категории DDE можно связать с данными, поступающими от внешних устройств, либо с объектами других пакетов, например ячейкой Excel.

Переменные могут объединяться в иерархические структуры (SuperTags), создавая логическую группу, их обработка осуществляется как обработка родственных данных, что значительно сокращает общее время обработки.

Переменные могут получать свои значения от контроллеров по различным протоколам — DDE, NetDDE, SQL, сетевым протоколам. Основным стандартом взаимодействия между программными компонентами системы SCADA стал OPC (OLE for Process Control): сервер диспетчерского управления обращается к OPC-серверу на контроллере или сетевом шлюзе и в ответ получает значение переменной.

В настоящее время наблюдается тенденция применения интегрированных программных продуктов, включающих в себя как системы SCADA, так и системы управления более высокого и низкого уровня. Компании Wonderware и Adastra поставляют в составе своих пакетов средства программирования контроллеров и средства управления производством. Хотя их возможности ограничены, они позволяют создать комплексную ИУС без применения дополнительных продуктов третьих фирм.

Для взаимодействия с вышестоящими уровнями всегда реализуются средства передачи информации без возможности управления. Обычно для этого используется протокол HTTP, и клиент с помощью WEB-браузера обращается к WEB-серверу (в TraceMode он называется WEB-активатор) на сервере SCADA или сетевом шлюзе.

Одной из важнейших функций системы SCADA является обязательное протоколирование информации, которая часто имеет и юридическую силу. Для этого используются локальные и глобальные регистраторы, по информации которых с помощью генератора отчетов формируются разнообразные отчеты с воспроизведением ситуации, которая наблюдалась в системе в заданное время (Playback архива).

Таким образом, на уровне диспетчерского управления мы получаем достаточно сложную систему, состоящую из нескольких, как правило, дублированных серверов и некоторого количества рабочих станций оперативного персонала.

Примером реализации ИУС на базе инструментальной системы SCADA Trace Mode является система автоматизированной противопожарной защиты энергоблоков № 3 и № 4 Тяньваньской АЭС (Китай, провинция Цзянсу), введенная 5 февр. 2019 г. в промышленную эксплуатацию на базе ОС Astra Linux Special Edition. Система автоматического управления противопожарной защиты контролирует более 850 000 сигналов от различных устройств АЭС, может служить до 50 лет в агрессивных средах без дорогостоящего обслуживания и перезаправки, выдерживает сейсмические воздействия до 8 баллов и температуру до –60 °С.

Еще одним направлением уровня диспетчерского управления является объектная видеоаналитика реального времени, использующая системы искусственного интеллекта на основе нейрокомпьютерных систем и машинное зрение для обработки в реальном времени значащих данных из видеопотоков. Объектная видеоаналитика приходит на смену традиционному видеонаблюдению с оператором — диспетчером, который после 30 мин просмотра обычно теряет способность замечать детали. В рамках объектной видеоаналитики на конечном аппаратно-программном устройстве запускаются нейронные сети, которые решают задачи распознавания номеров и моделей автомобилей, идентификации лиц и голоса, анализа текстов и др. Для непрерывной работы нейронной сети на конечном устройстве (инференс), могут использоваться специализированные программируемые логические матрицы (FPGA), графические процессоры (GPU) или процессоры общего назначения (CPU).

Типичное решение на CPU — это сервер с материнской платой на один-два слота для процессоров, например Intel Xeon, выполняющих инференс на логических ядрах процессора, число которых обычно равно количеству физических ядер либо вдвое больше при наличии технологии Hyper-threading.

Инференс на GPU более эффективен, чем инференс на CPU, за счет многоядерной архитектуры процессора и наличия высокоскоростной памяти. Одним из основных недостатков таких вычислений является высокая стоимость решения, которая складывается из цены видеокарт

и общей обвязки. Кроме того, необходима адаптация существующих сетей различных топологий для использования GPU.

Совместное использование CPU и GPU для одновременного выполнения инференса на CPU и GPU позволяет на лету обрабатывать до 80 потоков видео качества FullHD на скорости 15 FSP (frames per second), минимально необходимой для обеспечения плавности движения. Передача данных изображения происходит по протоколу RTSP (Real Time Streaming Protocol). Максимальная скорость инференса для обработки 15 фреймов в секунду и видеопотоков с 10 видеокамер может быть достигнута уже на стандартном компьютере Intel NUC8i5BEK с процессором Core i5 8259U [2].

Одним из успешных решений ИУС задач объектной видеоаналитики стала серия устройств OutDoor Box, представляющих собой промышленный одноплатный компьютер, способный в реальном времени выполнять инференс нейроннных сетей непосредственно рядом с источником данных. Рассмотрим параметры устройства OutDoor Box Micro. Оно представляет промышленный ПК на основе Up Core UP-CHT01 (CPU Intel Atom x5), позволяющий в реальном времени обрабатывать два транспортных потока при скорости движения до 250 км/ч. Основное его применение состоит в детектировании нарушений правил дорожного движения. Система работает по протоколу RTSP, а в случае обрыва беспроводного соединения, продолжает функционировать, сохраняя события в памяти и позволяя запросить их после восстановления связи по REST API [2].

Отметим, что на уровне диспетерского управления данными используются:

- специализированная аппаратура или аппаратура общего назначения;
- специализированные сетевые технологии или технологии общего назначения;
- специализированное системное программное обеспечение (ОС РВ) или ОС общего назначения;
- специализированное прикладное программное обеспечение (инструментальные системы диспетчерского управления).

На уровне *оперативного управления производством* (MES) (см. рис. 1) решаются типовые задачи управления предприятием с применением систем общего пользования. Характерные временные интервалы оперативного управления предприятия составляют минуты, часы, смены.

Задач реального времени здесь нет. В то же время многие подразделения и службы предприятия нуждаются в получении оперативной информации о состоянии управляемого объекта.

Покажем задачи уровня оперативного управления производством на примере электростанции. Ремонтные службы должны проводить регламентные работы на управляемом объекте. Для этого они должны согласовывать свою работу с ситуацией на управляемом объекте и, следовательно, нуждаются в информации о нем. Особенно это важно, когда ремонт на управляемом объекте нужно провести не по регламенту, а по состоянию оборудования. Топливные службы контролируют подачу топлива на энергоблок и должны планировать объемы и темпы его подачи. В этом случае необходимо рассчитать требуемое количество вагонов с углем, привести уголь со склада, превратить его в сырье нужной пылевидной консистенции, подать его на котельный агрегат. Расчеты и планирование также осуществляется в соответствии с актуальной информацией об управляемом объекте. Службам распределения электроэнергии и тепла актуальная информация об управляемом объекте позволяет рассчитать объемы и темпы поставки готовой продукции и т. д. Информация об управляемом объекте нужна практически всем подразделениям предприятия. В то же время не предусматривается управление объектом. Информация передается только в одну сторону — с диспетчерского уровня на уровень оперативного управления производством. На данном уровне используется обычная вычислительная сеть предприятия, информация поступает и обрабатывается на серверах и рабочих станциях АС предприятия. Речь идет о подсистемах АС предприятия, связанных по данным с управляемым объектом.

Таким образом, на уровне оперативного управления производством используются:

- аппаратура общего назначения;
- сетевые технологии общего назначения;
- системное программное обеспечение общего назначения;
- прикладное программное обеспечение общего назначения (АС корпоративного уровня).

На уровне *планирования ресурсов предприятия* (см. рис. 1) решаются типовые задачи планирования ресурсов (MRP, ERP) с использованием АС общего пользования.

Рассмотрим задачи уровня планирования ресурсов на том же примере электростанции. Кроме производственных проблем, управле-

ние электростанцией должно заключать договоры с добывающими предприятиями, отслеживать и контролировать поставки, разрешать возникающие при этом конфликты, чтобы организовать производственный процесс. Далее полученную электроэнергию необходимо поставить на оптовый рынок электроэнергии. Для этого необходимо заключить договоры с энергосетевыми предприятиями, отслеживать и контролировать поставки, разрешать возникающие при этом конфликты. В решении данных вопросов персонал электростанции использует подсистемы АС предприятия. Типичные временные интервалы составляют дни, недели, месяцы, годы. На этом уровне осуществляется долговременное планирование. Задачи реального времени здесь также отсутствуют. Однако заинтересованным подразделениям необходима информация о текущем, прошлом и прогнозируемом состоянии управляемого объекта.

Сделаем еще несколько замечаний относительно совместной работы всех уровней, использующих сетевые технологии. На уровне управления вводом-выводом используется специализированная промышленная сеть, обеспечивающая взаимодействие контроллеров, а на уровне диспетчерского управления может быть применена либо промышленная сеть, либо сеть общего назначения, объединяющая серверы и рабочие станции диспетчерского управления. Объединение сетей разных уровней не является целесообразным, т. к. может привести к снижению уровней надежности в управлении и безопасности объекта управления в целом. Даже в том случае, если на обоих (I и II) уровнях используются промышленные сети, они, как правило, реализованы по различным технологиям. Например, фирма Siemens рекомендует в качестве сети уровня II использовать промышленную сеть PROFIBUS, имеющую RS-485 и маркерный код доступа, а для построения сети уровня III — промышленную сеть PROFINET, являющуюся промышленным вариантом коммутируемого Ethernet. Сети PROFIBUS и PROFINET существенно отличаются друг от друга и могут быть соединены только через шлюз.

При применении типового протокола OPC, для взаимодействия контроллеров и системы SCADA, сервер OPC может располагаться либо как обычно — на контроллере, либо на шлюзе. Соответственно в первом случае должен быть реализован обмен данными между клиентом (например, SCADA) и аппаратурой (контроллерами, модулями ввода-ввода и др.) в реальном времени. Во втором случае информа-

ция, полученная от контроллеров, должна быть транслирована на III уровень. При использовании продуктов, реализующих IEC-61131, как например ISaNet в системе ISaGRAF, контроллер получает выполняющуюся на нем программу от системы SCADA. В этом случае шлюз должен пропускать такого рода сообщения.

Верхние уровни ИУС (IV–V) реализуются на базе обычной сети предприятия, которая соединяется с сетью диспетчерского управления (уровень III) через шлюз. Объединять эти сети категорически нельзя, чтобы исключить любую возможность управления объектом из сети предприятия. В то же время подсистемы АС предприятия, работающие с объектом, должны получать информацию о его состоянии. Для этого наиболее целесообразным является размещение WEB-сервера на шлюзе. WEB-сервера будут отображать актуальное состояние объекта, предоставлять отчеты о его состоянии и архивные данные. Однако маршрутизация любых пакетов через этот шлюз должна быть полностью исключена.

Сеть предприятия может иметь соединение с сетью общего пользования (сетью Internet). Для обеспечения безопасности, в соответствии с нормативными документами многих организаций и отраслей, такое соединение должно быть исключено. В этом случае приходится выделять отдельный изолированный от сети предприятия сегмент сети, соединенный с сетью общего пользования и используемый подразделениями предприятия, для которых такое соединение необходимо.

1.3. Концепция интернета вещей в информационно-управляющих системах

Системы интернета вещей появились достаточно давно, но в то время они так не назывались. Например, в 1982 г. студенты одного из американских вузов подключили автомат с кока-колой к локальной сети университета и, сидя за партами, проверяли, есть ли в автомате холодная кола, чтобы лишний раз не спускаться. Технологий беспроводной связи в то время еще не было, все соединялось проводами. В 1993 г. была выпущена умная кофеварка, которая сообщала своим владельцам о том, что кофе закончился.

Изобретателем концепции и термина «интернет вещей» (Internet of Things, IoT) является маркетолог Кевин Эштон (Kevin Ashton). Работая в компании Procter & Gamble в 1999 г., он обратил внимание, что помада одного цвета заканчивается быстрее других, и ее невозможно купить, т. к. новая партия поступит только в следующий отчетный период. Тогда ему пришла в голову мысль: а почему бы не отслеживать заканчивающийся товар и через интернет автоматизировать процесс закупки?

Отличие систем интернета вещей от других автоматизированных систем управления заключается в возможности объединить разные сети и данные, поступающие от разных источников, создавая новые возможности автоматизации.

Системы, использующие концепцию интернета вещей, можно разделить на 4 категории: мониторинг, управление, оптимизация, автоматизация (рис. 2).

Чем выше категория, тем функциональнее и «умнее» система. Категория «мониторинг» предполагает только сбор информации об объекте, «управление» позволяет автономно управлять объектами, включать или выключать механизмы и устройства. На уровне «оптимизация» система обладает интеллектуальными возможностями, может анализировать ситуацию и события. Категория «автоматизация» имеет полностью автономную умную систему управления, она имеет все уровни ИУС.

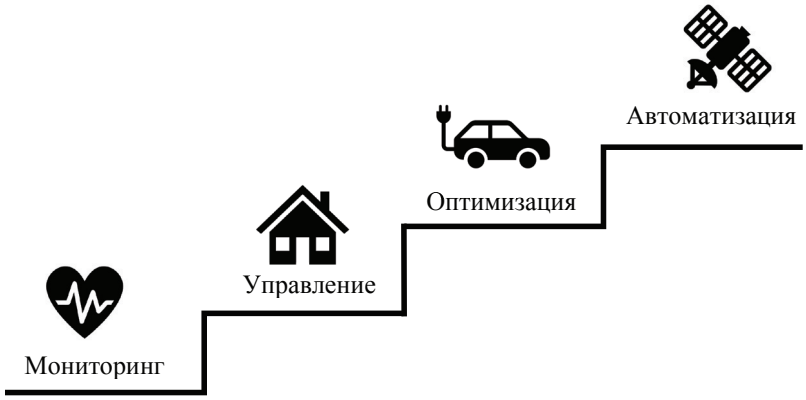

Рис. 2. Категории задач интернета вещей

Сферы применения систем IoT очень широки (рис. 3). Это и умный город (счетчики, мусорные баки, светофоры, транспортный шеринг,

уличное освещение города), и умное производство, умные дома, умное животноводство, умная медицина и т. д.

Рис. 3. Сферы применения интернета вещей

Состав самой простой системы интернета вещей (рис. 4) включает в себя: оконечные устройства (маленькие низкопотребляющие устройства, способные передавать данные), сеть (любая среда передачи данных), сервер сети (хранение данных), приложение (программа для визуализации и (или) управления данными). Опишем функционал структурных уровней интернета вещей:

- IoT-устройства (оконечные устройства), которые собирают показания с датчиков и выполняют физические действия. Могут быть персональными, носимыми и встроенными;
- шлюзы получают информацию от устройств и передают им команды выполнения действий. Как правило, они представлены аппаратным маршрутизатором или программным обеспечением и используют разные протоколы;

- сервер сохраняет, обрабатывает и анализирует показания датчиков. Он может быть реализован на базе виртуального сервера, реальной машины или через облако;
- клиентскую часть, которая реализуется через мобильное или веб-приложение и обеспечивает доступ к данным устройств и наглядному представлению результатов анализа.

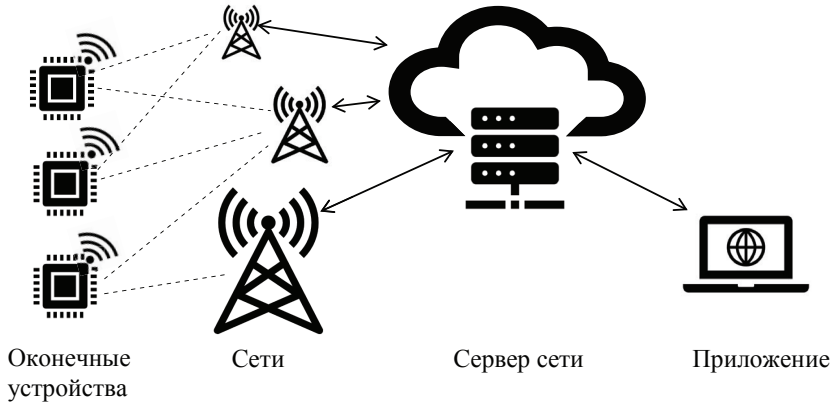

Рис. 4. Структура системы интернета вещей

Сектор стандартизации электросвязи Международного союза электросвязи предлагает следующее определение **интернета вещей** — это глобальная инфраструктура для информационного общества, которая обеспечивает возможность предоставления более сложных услуг путем соединения друг с другом (физических и виртуальных) вещей на основе существующих и развивающихся функционально совместимых информационно-коммуникационных технологий.

1.4. Архитектурные решения для построения систем в концепции интернета вещей

Традиционно системы IoT реализуются в архитектуре клиент — сервер (рис. 5). Конечным элементом системы является микроконтроллер (МК) с набором датчиков (Д) и исполнительных устройств (ИУ).

Микроконтроллер через сеть Интернет отправляет и принимает данные от web-сервиса. Развитие системы неминуемо усложняет программное обеспечение микроконтроллера, т. к. от последнего требуется реализация всех сетевых уровней и наличие алгоритмов обработки сбоев.

Отметим недостатки систем, реализуемых в архитектуре клиент — сервер:

- отсутствие асинхронного интерфейса;
- отсутствие буферизации;
- статическая маршрутизация;
- высокие затраты при модернизации;
- наличие жестких связей и зависимостей;
- ограниченный набор коммуникационных технологий;
- сложность программного обеспечения на стороне клиента.

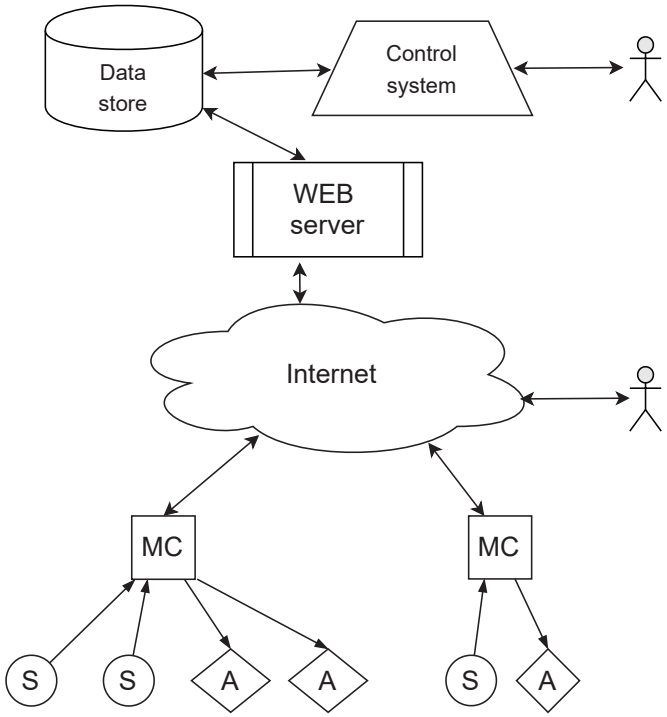

Рис. 5. Система IoT в архитектуре клиент — сервер

Эти недостатки сокращаются при использовании сервис-ориентированной архитектуры (SOA), ключевым технологическим элементом которой является сервисная шина предприятия ESB.

Структурная схема системы IoT, построенная в архитектуре ESB, представлена на рис. 6. Конечным элементом системы является микроконтроллер (МК) с набором датчиков (Д) и исполнительных устройств (ИУ). Программное обеспечение микроконтроллера реализует:

- чтение показаний датчиков;
- выполнение команд управления;
- обмен данными с адаптером в синхронном режиме.

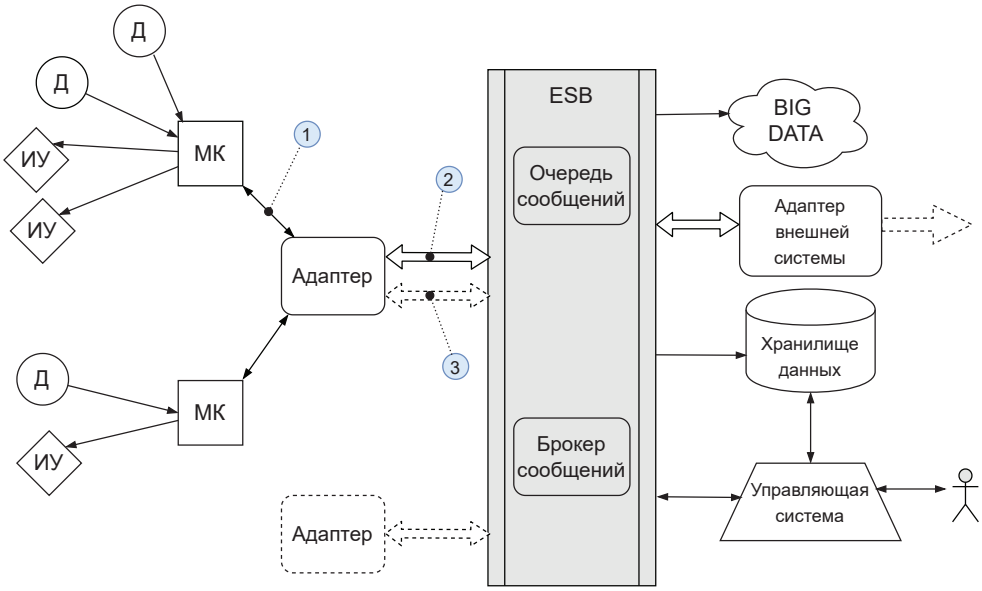

Рис. 6. Сервис-ориентированная архитектура ИУС

Адаптер является промежуточным звеном, которое осуществляет взаимодействие с микроконтроллерами по протоколу низкого уровня исходя из географической удаленности и технических возможностей связи. В задачи адаптера входит буферизация сообщений, трансформация в формат шины данных и отправка на сервер очереди сообщений. Адаптер имеет возможность переключения на альтернативный канал передачи данных. В его функции входит мониторинг состояния подключенных к нему узлов и сигнализация состояния системы. Программное обеспечение адаптера настраивается в зависимости от функционального назначения вверенной ему подсистемы и может быть перенастроено без внесения изменений в другие части системы.

Сервис-ориентированная архитектура избавляет конечные узлы от задач коммуникационного взаимодействия, т. к. эта функция возложена на адаптер. На рис. 6 цифрой 1 обозначен канал передачи данных низкоуровневого протокола, цифрой 2 — основной канал связи с системой шифрования данных, 3 — резервный канал связи.

Шина данных обеспечивает хранение, диспетчеризацию и трансформацию сообщений. Таким способом обеспечивается интеграция с различными системами, в т. ч. внешними. Без модификации компонентов системы IoT возможна динамическая настройка маршрутов и подключение дополнительных узлов.

Существуют готовые открытые программные средства и библиотеки, реализующие функционал шины данных. На промышленном уровне целесообразно использовать коммерческие решения, такие как IBM WebSphere ESB, Oracle Enterprise Service Bus, SAP Process Integration и др. Для целей гражданского и бытового применения систем IoT вполне достаточно возможностей таких продуктов, как Apache Camel, JBoss ESB и т. п.

Контрольные вопросы для самопроверки

1. Дайте определение ИУС. Укажите основные задачи, решаемые ИУС.
2. Приведите примеры ИУС в нефтегазовой промышленности, энергетике и на транспорте.
3. Какую роль играют ИУС в научных исследованиях? Какие задачи решает ИУС в научных исследованиях?
4. Укажите принципиальные особенности ИУС, отличающие ее от обычной АС.
5. Что такое работа в реальном времени? По каким параметрам она оценивается?
6. Дайте определение жесткой и мягкой системы реального времени. Приведите их примеры.
7. В чем состоит иерархический принцип построения ИУС. Опишите типичную пятиуровневую иерархическую структуру.
8. Какова особенность первого уровня пятиуровневой иерархической структуры ИУС?

9. Какие задачи решают контроллеры ИУС? Какие функции реализуют контроллеры? Каким образом обеспечивается работа контроллера в реальном времени?

10. Определите задачи диспетчерского уровня управления и сбора данных ИУС. Какое аппаратное и программное обеспечение используется на этом уровне?

11. На каком уровне реализуются алгоритмы управления объектом?

12. Что такое система высокой готовности? Каким образом достигается коэффициент готовности «пять девяток»?

13. Какое прикладное программное обеспечение используется на диспетчерском уровне управления и сбора данных ИУС?

14. Что такое переменные управления? Какую роль они выполняют при создании графического пользовательского интерфейса и интерфейса оператора? Укажите протоколы, по которым переменные получают свои значения от контроллеров.

15. Укажите функции SCADA. Приведите примеры использования SCADA.

16. Какие задачи решаются на уровне оперативного управления производством? Укажите особенности аппаратного и программного обеспечения этого уровня ИУС?

17. Кому принадлежит термин «интернет вещей» и какие задачи решают такие системы?

18. Приведите примеры систем интернета вещей. Объясните отличие таких систем от типичных АСУ ТП.

19. Какие части входят в ИУС интернета вещей?

20. Раскройте определение ИУС, построенных в концепции интернета вещей.

2.1. Концепция открытых систем

Концепция открытых систем является базовой в развитии современных компьютерных технологий. Современные ИУС разрабатываются как открытые системы (Open Systems), которым свойственны расширяемость или масштабируемость (extensibility/scalability), мобильность или переносимость (portalility), интероперабельность, или способность к взаимодействию с другими системами, (interoperability) и дружественность к пользователю, включая простоту управляемости (driveability).

Указанные свойства, взятые по отдельности, встречались и у предыдущих поколений ИУС, однако в рамках концепции открытых систем они реализуются комплексно.

Концепция открытых систем развивается с 1980-х гг. Она заложена фирмой Redstone (Team Redstone), выдвинувшей принцип NDI[1] (Non-Developmental Item), что означает принцип неразрабатываемых заново технологий [3]. Этот принцип основывался на экономической (рыночной) релевантности компьютерных продуктов и услуг. В конце XX столетия эта тенденция совпала с мнением многих инженерных сообществ о целесообразности унификации способов и протоколов передачи данных между всеми компонентами и уровнями компьютерных систем — от оперативной памяти и процессоров до всех видов периферии.

Применение принципа NDI привело к формированию фактически трех линеек аппаратно-программных продуктов: COTS, MOTS и ROTS [3] — и дальнейшему построению ИУС по модульному принципу с использованием этих продуктов.

COTS (commercial off the shell) — это коммерчески готовые и коммерчески доступные программные продукты. Они представляют собой упакованные решения, которые в дальнейшем адаптируются для

[1] Аббревиатура NDI является достаточно распространенной и встречается совершенно в другом значении.

удовлетворения потребностей организации-покупателя. Продукты COTS не делаются на заказ. У них нет стадии ввода в эксплуатацию. Федеральный регламент закупок США (FAR) определил «COTS» как формальный термин для коммерческих товаров и услуг, которые доступны на коммерческом рынке и могут быть куплены и использованы по государственному контракту. Например, фирма Microsoft является поставщиком программного обеспечения COTS. Товары COTS являются альтернативой заказному программному обеспечению или разовым индивидуальным разработкам.

Хотя продукты COTS можно использовать «из коробки», на практике продукт должен быть интегрирован в некоторую индивидуальную систему, что может привести к расширению его функционала. Последнее может послужить причиной конфликтной ситуацией и отказа в дальнейшей поддержке и обслуживании.

В целом такой подход привел к резкому снижению цен на программные продукты, сделал их достаточно доступными и послужил именно тем толчком в бурном развитии компьютерных технологий, которые существенно изменили нашу жизнь за последние два десятилетия.

Отметим, что бурное развитие популярного продукта сопровождается мошенничеством. По оценкам института SANS, в дек. 2012 г. только 14% компаний, работающих с программными продуктами COTS, провели проверки кибербезопасности каждого коммерческого приложения, введенного собственными силами. В то же время более половины компаний не проводили никакого анализа кибербезопасности, полагаясь только на репутацию поставщика COTS (25%) и соглашения о юридической ответственности (14%).

Особую опасность представляет оборудование, планируемое к использованию в реальном времени в медицине. Значительный процент программного обеспечения COTS был идентифицирован как SOUP (программное обеспечение с неизвестной родословной или программное обеспечение с неизвестным происхождением). Такая ситуация может через несколько лет привести к тому, что при оказании медицинских услуг сбои в программных компонентах могут стать системными сбоями в самом устройстве.

Две линейки MOTS (Military off-the-shelf — готовые изделия военного исполнения) и ROTS (Rugged off-the-shelf — готовые изделия повышенной надежности) — предназначены для выпуска программных продуктов повышенной надежности, для работы в сложных ситуациях.

Модули MOTS имеют эксплуатационный температурный диапазон –55 ...+105 °С; в них применяются полупроводниковые компоненты повышенной надежности с температурным диапазоном –55 ...+125 °С; используется кондукционное охлаждение (с помощью теплоотводов). Они имеют самые жесткие механические характеристики, поэтому возможно их применение в подвижных объектах.

Для этой линейки изделий разработаны дополнительные международные стандарты: MIL-STD-704 (для авиации), MIL-STD-1275 (для наземной техники), MIL-STD-1399 (для флота), MIL-STD-810 (для разных сред).

Модули ROTS имеют типовой эксплуатационный температурный диапазон –40...+85 °С. Они выполняются из полупроводниковых компонентов повышенной надежности. На них может быть разработана стационарная техника стоечного или настольного исполнения, способная функционировать в тяжелых условиях, в т. ч. вне помещений. Применение в подвижных объектах возможно только при дополнительном демпфировании.

Необходимыми условиями успешного применения рассмотренных линеек модулей являются:

- наличие и соблюдение стандартов на интерфейсы;
- рыночная доступность программно совместимых модулей разных исполнений и разных производителей.

2.2. Интерфейсы открытых систем

При проектировании аппаратуры информационно-управляющих систем, из-за обычно большого количества каналов ввода-вывода, приходится (как минимум на уровне управления вводом-выводом) применять модульный принцип ее построения. Поэтому строятся стандартизированные магистрально-модульные системы (ММС), состоящие из конструктивно обособленных модулей, связанных (обычно локальной) магистралью обмена информацией. В качестве механического стандарта для размещения этих модулей наиболее широко применяется «Евромеханика» по IEC 60297 (ГОСТ 28601) [4–6].

«Евромеханика» (*англ.* Eurocard — европейская плата, называют также 19-дюймовым стандартом) — стандартный конструктив, позволяющий единообразно разместить в субблo-

ке и после этого в стойке разнородные модули [4, 5]. Впервые такой конструктив был применен в 1920-х гг. производителем техники связи компанией Bell для размещения оборудования модульных автоматических телефонных станций (рис. 7).

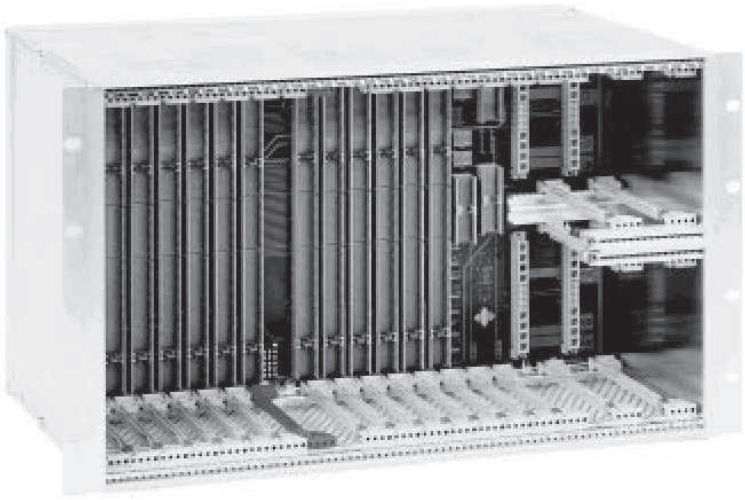

Рис. 7. Субблок с европлатами двух форматов: 3U (100х160 мм) и 6U (233х160 мм)

Конструктив «Евромеханика 19 дюйм» включает набор требований к механическому исполнению модулей, что позволяет устанавливать его в стандартные крепления [5]. Понятие конструктива не накладывает прямых ограничений на функциональные, электрические или какие-либо еще свойства. Собственно модуль оформляется в виде платы с панелью управления на лицевой стороне и разъемами на задней. На передней панели модуля могут находиться разъемы для подключения оборудования, элементы управления или индикации. Модули устанавливаются в субблок вертикально — вдвигаются туда по направляющим, каждая в свой слот. Слоты, в которых модули отсутствуют, могут закрываться заглушками. Достаточно часто субблок также называют крейтом (*англ.* crate, одно из значений — ящик для хранения стекла с направляющими). Будучи вставленным в субблок, разъемы задней части модуля совмещаются с разъемами задней стенки крейта. Используются разъемы типа DIN 41612 или его аналога СНП59.

Рекомендуемое расположение соединителей на кросс-плате для основных типов разъемов регулируется стандартом IEC 603-2 (DIN 41612).

Субблоки своими фланцами закрепляются в стойке. Стойка содержит вертикальные швеллеры с отверстиями, за которыми могут быть установлены резьбовые шпонки (сухари), к которым и привинчиваются фланцы субблоков.

Стандарт предусматривает возможность установки дополнительных элементов, не мешающих функционированию конструкции (ручки, защелки и т. п.).

Стандарт определяет формат и типовой размер используемых печатных плат, форму, профиль, расположение и допуски на элементы крепления (требования распространяются на платы и крейты).

Стандарт рекомендует общее расположение разъема на плате.

Все основные размеры стандартизированы. Высота субблока должна быть кратна специальной монтажной единице U (Unit, 1.75“, 44.45 мм). Наиболее распространены субблоки с платами высотой 3U, 6U и 9U. С учетом запаса на направляющие элементы стандартная высота плат определена как 100, 233.5 и 366.7 мм соответственно.

По глубине места платы могут иметь размер (длину) от 100 мм, увеличиваясь с шагом 60 мм. Типовые значения — 100, 160, 220, 280, 340 и 400 мм.

Ширина слота для плат задается в монтажных единицах HP (Horizontal Pitch) и составляет 0,2“, что равно 5.08 мм. Она соответствует максимальной толщине платы и компонентов на ней. Обычная ширина субблока для установки плат — 84 HP.

Предпочтительные размеры плат для установки в субблок имеют высоту 3U: 100×160 мм и 100×220 мм; или 6U: 233.35 ×160 мм и 233.35×220 мм. Посадочный размер в стойке — 465.1 (±1.6) мм, ширина субблока — 482.6 мм. Расстояние между вертикальными держателями передних панелей в стойке около 450 мм [6, 8].

Для обозначения степени защиты оборудования от воздействий окружающей среды используется система кодов IP согласно IEC 529. Степень защиты кодируется в виде IP XY, где X — степень защиты от твердых тел и пыли, а Y — степень защиты от влаги (табл. 1). Например, некоторое устройство объектного видеонаблюдения, обрабатывающее в реальном времени транспортные потоки, находится в антивандальном корпусе со степенью пыле- и влагозащищенности IP66 и способно работать в диапазоне –40...+50 °C.

Таблица 1

Обозначение степени защиты компьютера от воздействий окружающей среды (коды IP)

Степень защиты	Защита от твердых тел, X	Защита от влаги, Y
0	Защита отсутствует	Защита отсутствует
1	Защита от тел диаметром более 12 мм	Защита от вертикально падающих капель воды
2	Защита от тел диаметром более 12 мм	Защита от капель воды, падающих под углом до 15° от вертикали
3	Защита от тел диаметром более 2,5 мм	Защита от капель воды, падающих под углом до 60° от вертикали
4	Защита от тел диаметром более 1 мм	Защита от брызг воды, попадающих на оболочку с произвольного направления
5	Проникновение пыли не приводит к нарушению работоспособности изделия (системы)	Защита от струи воды, выбрасываемой с произвольного направления
6	Проникновение пыли полностью исключается	Защита от сильной струи воды, выбрасываемой с произвольного направления
7	Не предусмотрено	Защита от проникновения воды при погружении на глубину 150 мм
8	Не предусмотрено	Защита от проникновения воды при погружении на глубину, определяемую изготовителем

Значительная часть аппаратуры АС сосредоточена в центрах обработки данных (ЦОД). Разработкой открытых стандартов и архитектур занимается некоммерческая организация под названием Open Compute Project (OCP). Она выросла из проекта перепроектирования ЦОД Facebook.

Основной задачей разработчиков OCP является повышение эффективности, снижение капитальных и текущих затрат для инфраструк-

туры крупномасштабных ЦОД. Разработчики OCP стремятся достичь универсальности и простоты масштабирования инфраструктуры и вычислительных мощностей и при этом минимально влиять на его работоспособность, поддерживать режим горячей замены и задействовать минимальное количество обслуживающего персонала, а также автоматизировать мониторинг потребляемых ресурсов и контролировать статистику сбоев.

Одним из основных проектов OCP являлась разработка стандарта серверных стоек Open Rack. С сохранением наружной ширины и глубины, осуществлена установка более широких шасси шириной 537 мм (около 21 дюйма), что позволило разместить больше оборудования в том же самом объеме и улучшить прохождение потока воздуха. Высота вычислительных шасси составляла 48 мм, что существенно больше обычной телекоммуникационной единицы 1U (юнита). В итоге было предложено привычные в IT-индустрии 19-дюймовые конструкции заменить 21-дюймовыми модульными стойками.

Объем рынка технологий OCP составил 1,16 млрд долл. в 2017 г., и ожидается дальнейшее расширение рынка, подкрепляемого развитием сетей 5G и потребностей в виртуализации.

Для передачи электрических сигналов и цифровой информации в магистрально-модульных системах (ММС) используются специализированные интерфейсы, обеспечивающие создание локальной системной магистрали субблока. Эти интерфейсы прошли долгий путь развития.

Первое поколение встраиваемых ИУС (NIM, CAMAC) имело модульную структуру с магистральной архитектурой (ММС), процессоры с 8-разрядной шиной и тактовой частотой до нескольких мегагерц [7–9]. Вертикальная установка модулей обеспечивала естественную вентиляцию. В универсальной информационной магистрали реализовался принцип унификации способов и протоколов передачи данных между всеми компонентами и уровнями компьютерных систем — от оперативной памяти и процессоров до всех видов периферии. Магистрально-модульные системы (ММС) в целом отличались невысокой производительностью и пропускной способностью.

На основе крейтового решения, интегрированного с магистралью и требуемым набором модулей, развивается гибкая магистрально-модульная система с высокой надежностью. Такая система способна ра-

ботать в широком диапазоне температур и влажности при наличии вибраций и пыли. Именно эти качества позволяют использовать ММС в тяжелых условиях промышленной автоматизации. Рассмотрим этапы развития магистрально-модульных систем (рис. 8).

Системы FASTBUS, а также FUTUREBUS представляют второе поколение ММС [7, 8]. VME и Compact PCI [10] — системы третьего поколения. Они успешно применяются по сей день. Четвертое поколение модульных систем с коммутируемой средой, например VPX, построены на основе наноэлектронных СБИС.

Магистрально-модульные системы с 32-разрядной параллельной шиной — VME [9], Compact PCI [10–14], FASTBUS [7–9] — стали основой развития систем высокой производительности с многопроцессорной структурой. В табл. 2 приведены характеристики базовых интерфейсов ММС.

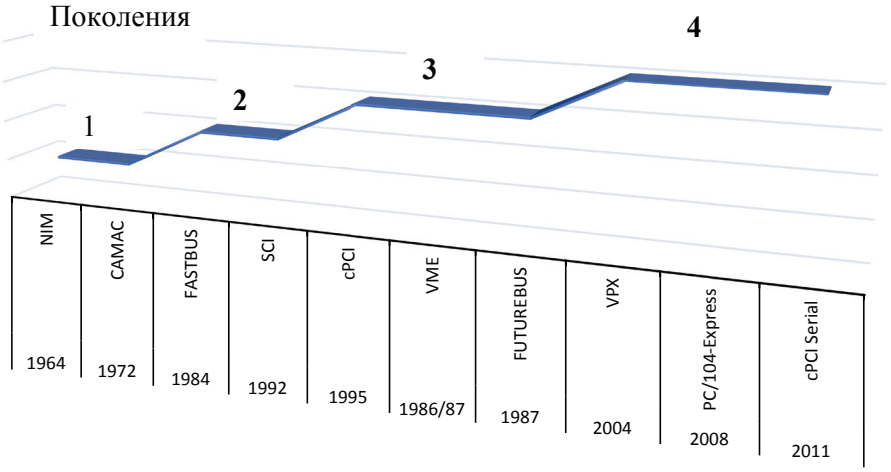

Рис. 8. Этапы развития магистрально-модульных систем

Таблица 2

Базовые стандартизованные интерфейсы

Характеристики	VME	Compact PCI	Fastbus
Стандарт	ANSI/IEEE 1014-1987 ANSI/VITA 1-1994	PICMG 2.0: CompactPCI (1999), 32-разрядная шина PCI	IEEE 960 34.340-91 *ФАСТБАС*

Продолжение табл. 2

Характеристики	VME	Compact PCI	Fastbus
Описание стандарта	Стандарт на компьютерную шину, первоначально разработанный для семейства микропроцессоров Motorola 68000 и нашедший применение для множества других приложений. Есть спецификация «горячей замены»	Системная шина, широко используемая в промышленной автоматике. Является совместимой с предшественниками. Есть спецификация «горячей замены»	Стандарт компьютерной шины и крейтовой системы, ориентированный на высокоскоростной сбор данных с параллельно работающих модулей. Состоит из одного или нескольких сегментов. Сегменты разделяются на крейтовые и кабельные
Развивающие организации	*Versa VMEbus International Trade Association*	Консорциум PICMG (сокр. от *англ.* PC Industrial Computer Manufacturers Group); PCI Special Interest Group	Объединенный комитет NIM/КАМАК Министерства энергетики США
Базовая процессорная платформа	*Motorola*	*Intel* Версия 2.1: Alpha, MIPS, PowerPC, SPARC	68 000; VAX 11/780; iAPX 432
Разрядность шины	16/32/64	32/64	32
Конструктив «Евромеханика»	3U, 6U, 9U	3U, 6U	Немного выше, чем другие типы крейтов
Максимальное количество слотов	21	8/16/21 (с использованием мостов)	26 (модули имеют разную толщину)
Пропускная способность	40, 80 Мбайт/с	133, 533 Мбайт/с. Обмен данными между модулями — 1 Гбит/с	80 Мбайт/с

Продолжение табл. 2

Характеристики	VME	Compact PCI	Fastbus
Передача данных	Операции чтение — запись выполняются асинхронно и без мультиплексирования, но есть и синхронные и мультиплексированные режимы. Передача данных ориентирована на поддержку многопроцессорной среды. Блочная передача данных; максимальная длина блока — 2048 байт	Синхронная мультиплексируемая	Сегменты используют 32-битную шину передачи данных. Осуществляется мультиплексирующая передача адресов и данных по одним и тем же проводникам. Может работать асинхронно с использованием протокола подтверждений, чтобы эффективно сочетать устройства различного быстродействия без предварительного учета скорости их работы. Система способна работать и синхронно без подтверждений при передаче блоков данных с максимальной скоростью
Арбитраж шины	Механизм арбитража шины похож на операции запроса и прерывания: поддерживает 7 уровней прерываний;	В стандарте PICMG 2.0 R3.0 мультипроцессинг не специфицирован. Одноуровневые прерывания. Процессорные модули CompactPCI весь обмен с периферийными модулями ведут через мосты PCI-PCI.	FASTBUS определяет алгоритм арбитража при одновременной попытке нескольких ведущих модулей захватить сегмент.

Окончание табл. 2

Характеристики	VME	Compact PCI	Fastbus
Арбитраж шины	любое устройство, имеющее право быть ведущим, может запросить управление шиной через одну из четырех линий запроса; схема выбора очередного ведущего происходит через арбитра; недостаток — устройство, которое ближе к арбитру, чаще владеет шиной	Системный модуль отличается от периферийного тем, что содержит арбитр запросов шины, обработчик прерываний шины, системный генератор, «прозрачный» мост PCI-PCI. Ему доступны все ресурсы системы. Другие процессорные модули, устанавливаемые в несистемные слоты, имеют в своем составе так называемый «непрозрачный» мост PCI-PCI	Ведущий модуль управляет передачей данных по сегменту, самостоятельно начиная и оканчивая ее или передавая соответствующие команды ведомым модулям. Каждый из ведущих модулей ввода-вывода последовательно осуществляет блочную передачу, заканчивает захват шины и передает управление следующему в цепочке модулю
Развитие стандарта	VMS, VXI, VPX, Open VPX	PCI-Express; COM Express, с PCI Serial	SCI (Scalable Coherent Interface), в русском переводе — РСИ (расширяемый связный интерфейс)

Основным препятствием дальнейшего развития магистрально-модульных систем VME и Compact PCI стала сама традиционная параллельная магистраль, разделяемая поочередно всеми модулями системы. Дальнейшее увеличение ее разрядности до 64 привело к значительному росту помех и наводок в параллельных линиях связи. В магистральных системах на основе PCI отдельные модули и устройства имели возможность возможность монополизировать всю шину и ограничить доступ к общим ресурсам. Еще одним фактором, играющим негативную роль в развитии магистрально-модульных систем с архитектурой на основе параллельной шины, являлось ограничение пропускной способности шины.

В основу дальнейшего развития магистрально-модульных систем был положен *принцип гибридного подхода* к архитектуре магистрали с включением быстрого последовательного сопряжения и коммутируемой связи модульных систем [7, 9, 16]. В результате объединения новых технологий последовательной связи с коммутацией пакетов (VXS) с традиционными электрическим и механическим стандартами VME64, в систему были добавлены многогигабитные коммутируемые последовательные соединения (VITA 41 и 46) и новые мезонинные модули (VITA 42) [9, 16]. Последовательные быстрые передачи по спецификации VITA 46 привели к необходимости поддержания скорости передачи до 6,25 Гбит/с и замене соединителей в традиционных системах VME на более высокочастотные. Расширенные системы VME сохраняют совместимость с традиционной шиной. Гибридный подход обусловлен также тем, что организация прерываний, арбитраж и обработка приоритетов в многопроцессорных системах лучше реализуются в параллельных структурах.

Архитектура современных модульных систем развивается в нескольких направлениях. Одним из них является построение коммутируемых модульных последовательных структур (PCI Express) с централизованным управлением коммутацией [13]; другим — использование топологии многосвязных сетей типа ASI (Advanced Switching Interconnect). Еще одним направлением развития систем является разработка перспективных модульных архитектур следующего поколения ATCA с топологией одиночной или двойной «звезды» или полносвязной сети [7].

Отметим также развитие малогабаритных встраиваемых модульных систем из набора процессорных и других модулей, объединяемых с использованием этажерочной компоновки. Примером такой компактной встраиваемой модульной системы является стандарт PC/104 [7–9]. Его первые модули были выполнены на базе 8-разрядной шины ISA. При переходе на PCI, пропускная способность нового стандарта PC/104+ достигала 132 Мбайт/с. Переход на коммутируемый интерфейс PCI Express обеспечил пропускную способность до 2,5 Гбит/с. Введение новых технологии разъемов, интерконнектов, электропитания, механической конструкции привело к созданию Compact PCI Serial (cPCIs) с пропускной способностью до 800 Гбит/с [12, 13, 15].

Стремительное развитие сетевых технологий и значительное увеличение пропускной способности Ethernet-каналов способствовали появлению нового инструментального стан-

дарта LXI (LAN eXtension for Instrumentation), который, наряду со стандартами PXI, VXI, AXI, позволяет разрабатывать современные средства автоматизации экспериментов и контрольных испытаний [16]. Промышленная сеть на основе Ethernet здесь используется в качестве системной магистрали. Принят ряд мер по обеспечению работы сети в реальном времени. Примером такой системы является LXI-система сбора данных аэродинамических испытаний. LXI-система обеспечивает прием входных данных одновременно по 96 каналам со скоростью 216 ксэмпл/с на канал, высокую точность синхронизации и запуска в реальном времени [12, 13, 16].

На рис. 9 представлено изменение пропускной способности магистрально-модульных систем от первых поколений ММС до современных систем. У каждого шага была своя цена. Полоса пропускания второго поколения была расширена за счет высокого энергопотребления и тепловыделения. Переход к третьему поколению ММС обусловлен переходом к КМОП-технологии. Развитие четвертого поколения обязано сетевым архитектурам на основе коммутируемых вычислительных сред с увеличением степени интеграции СБИС (система или сеть на кристалле).

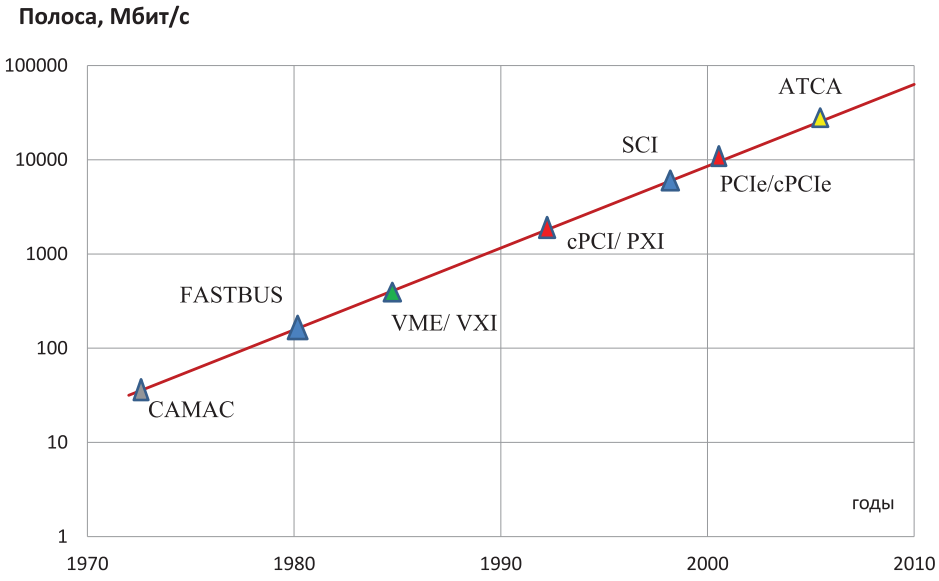

Рис. 9. Пропускная способность магистрально-модульных систем от первых поколений ММС до современных систем [7, 8, 9]

Четвертое поколение модульных систем указывает на перспективность развития масштабируемых серверных кластеров высокой производительности, эффективность которых будет достигаться в результате использования многоядерных модулей и создания широкополосной коммутируемой среды связи узлов.

2.3. Промышленные сети

Промышленная сеть (*англ.* Fieldbus — полевые шины) — это специализированная сеть передачи данных, связывающая различные датчики, исполнительные механизмы и промышленные контроллеры в автоматизированных системах управления технологическими процессами (АСУТП). Описывается многофункциональным стандартом IEC 61158 и рядом других [17].

Отличие промышленных сетей от сетей общего назначения уже рассматривалось ранее в подгл. 1.2. Промышленные сети являются узкоспециализированными, чаще всего имеют небольшой размер, ориентированы на работу в тяжелых условиях в составе систем реального времени. Применяются в ИУС (как минимум) на уровне управления вводом-выводом.

Международный стандарт IEC 61158 (Fieldbus for use in Industrail Control Systems) определяет 8 независимых и несовместимых коммуникационных технологий, среди которых наиболее распространены P-Net, CANopen и ProfiBus. Их описание и параметры представлены в табл. 3.

Таблица 3

Характеристики промышленных сетей на основе стандарта IEC 61158

Параметры	Название сети		
	P-Net	CAN	ProfiBus DP
Фирма-разработчик или развивающая организация	International P-NET User Organization (IPUO)	Bosch or CANopen Special Interest Group	Siemens AG or Profibus Network Organization PNO

Продолжение табл. 3

Параметры	Название сети		
	P-Net	CAN	ProfiBus DP
Дополнительные стандарты на шину и ее компоненты	EN 50170-1 (1997)	ISO 11898 (определяет только канальный уровень)	EN 50170, DIN 19245 Версии: Profibus DP Profibus FMS; Profibus
Общее описание стандарта	Физический уровень — RS-485, множественная кольцевая физическая топология, метод доступа с передачей виртуального маркера и фиксированной тактовой частотой	Физический уровень — ISO 11898, близкий к RS-485, топология — шина, метод доступа — CSMA/CR	Физический уровень RS-485, физическая топология — шина, маркерный метод доступа
Используемые уровни модели OSI	1 — физический уровень; 2 — канальный уровень; 3 — сетевой; 4 — транспортный; 7 — уровень приложений	CAN: 1 — физический уровень; 2 — канальный уровень. CAN open: 3 — сетевой; 4 — транспортный; 7 — уровень приложений	1 — физический уровень; 2 — канальный уровень; 7 — уровень приложений
Максимальная длина кабеля	1200 м без повторителей	5000 м	Длина сегмента без повторителей — 200–1200 м (при использовании повторителей до 4800 м)

Продолжение табл. 3

Параметры	Название сети		
	P-Net	CAN	ProfiBus DP
Средняя скорость передачи данных	76,8 Кбит/с Max — 500 Кбит/с	1 Мбит/с для кабеля длиной 40 м, 10 кбит/с — 5000 м	9,6 Кбит/с...12 Мбит/с
Посылка данных	Асинхронная передача в NRZ (no return to zero)	Синхронный механизм	Асинхронный механизм
Использование multimaster-системы	Multimaster-система может включать до 32 master-устройств на шину. Каждый сегмент может включать до 125 устройств	Не выделяет master-устройства или станции. Система состоит из узлов, между которыми происходит обмен информацией	Multimaster-система. В одном сегменте их общее количество ограничивается 32. Общее количество узлов во всей системе ограничено 127
Особенность доступа к шине или особенности арбитража	Доступ к шине осуществляется при помощи передачи виртуального маркера: master-устройства получают доступ к шине циклически, в порядке возрастания их адресов (данная схема была специально разработана для P-NET)	Тип доступа — Collision Resolving (CR, разрешение коллизии)	Маркерный метод доступа. Время цикла шины постоянно, оно не изменяется даже при добавлении или удалении узлов. Временные интервалы задаются не жестко. Master не обязан полностью использовать отведенное ему время, он может заранее передать свой маркер следующему узлу; 7-й уровень выполняет функцию связи типа клиент — сервер

Окончание табл. 3

Параметры	Название сети		
	P-Net	CAN	ProfiBus DP
Программирование и конфигурирование	VIGO	Библиотека python-can предлагает программные средства для пассивного мониторинга и активного контроля шины CAN. Для конфигурирования используется широкий набор драйверов и конверторов	Step 7 — пакет программирования контроллеров Simatic фирмы Siemens AG. Com Profibus –программа для конфигурирования сети Profibus. Функции Step 7 позволяют оптимизировать соединение с устройствами семейства Simatic S7

В таблице представлена только модификация Profibus DP. Profibus PA используется для взрывопожароопасных производств и отличается по физическим и канальным уровням (спецификация IEC 61158 с фиксированной тактовой частотой и питанием по кабелю передачи данных). Модификация Profibus FMS более не применяется и заменяется ProfiNet (на базе коммутируемого промышленного Ethernet) [18].

Поскольку промышленные сети, как правило, работают в системах реального времени, крайне важными являются временны́е характеристики этих технологий — задержки и их флуктуации [22]. Возможность получения предсказуемых значений этих параметров определяется прежде всего используемым методом доступа.

Использование маркерного метода доступа (Profibus и P-Net) гарантирует детерминированное поведение системы. Метод доступа с предотвращением коллизий (CSMA/CA), позволяющий выстроить жесткую последовательность приоритетов станций, также гарантирует предсказуемое поведение системы. Реализация третьего уровня модели ISO/OSI в сегментированной сети P-Net предоставляет возможность организовать статическую или динамическую маршрутизацию [17, 23].

Стремительный рост производительности управляющих устройств привел к тому, что традиционные промышленные сети стали узким местом ИУС [23, 24]. Одним из факторов, уменьшающим производительность полевых шин, стала многоуровневая архитектура управления, состоящая из нескольких подчиненных систем: текущего управляющего задания, полевой шины и, возможно, локальных шин расширения в системах ввода-вывода или собственного цикла микропрограммы периферийного устройства. Именно это стало приводить к увеличению времени реакции системы в 3–5 раз по сравнению с временем управляющего цикла.

Очередным шагом в развитии промышленных сетей стало развитие стандартов Real-Time Ethernet для передачи данных в сетях реального времени и реализация этих стандартов в коммутаторах Ethernet, предназначенных для использования в реальном времени [23].

Привлекательность стандарта Ethernet для промышленных сетей связана прежде всего с его высокой пропускной способностью, широким распространением и продолжением успешного развития этой технологии. Применение коммутаторов с дуплексными портами и подключением одной станции на порт позволяет избавиться от коллизий, приводящих к непредсказуемым задержкам. Это было реализовано в Ethernet-совместимых протоколах реального времени Ethernet/IP, ProfiNet, EtherCAT и Powerlink на основе технологий коммутации. Кроме того, отметим использование стандарта IEEE1588, позволяющего синхронизовать таймеры абонентов сети с высокой точностью [23].

Ethernet/IP (Ethernet Industrial Protocol) представляет открытый стек протоколов, в котором первые четыре уровня (физический, канальный, сетевой и транспортный) используются без изменений, а на прикладном уровне введен специальный объектно ориентированный протокол CIP (Common Industrial Protocol). Его работа в режиме реального времени обеспечивается специальным расширением протокола CIP — CIPSync, основанном на протоколе временной синхронизации IEEE 1588. Детерминированность сети определяется следующими механизмами, направленными на удаление любого постороннего трафика:

- каждый сегмент при обмене данными в реальном времени должен находиться в собственной виртуальной сети VLAN;
- функции резервирования линий связи (STP — Spanning Tree Protocol и его разновидности) должны быть отключены;

- каждый сегмент должен иметь топологию звезды с управляемым коммутатором второго уровня в режиме IGMP Snooping для фильтрования паразитных широковещательных сообщений в сети. Применяют коммутаторы с дуплексными портами и подключением одного узла на порт. Коллизии в такой сети отсутствуют;
- применяются коммутаторы третьего уровня, позволяющие отфильтровывать нежелательные пакеты данных с истекшим временем жизни, например при трансляции IP-видеосигнала по круговым маршрутам.

Применяется также EtherCAT (Ethernet for Control Automation Technology) — протокол, разработанный компанией Beckhoff для сетей Ethernet реального времени [23, 24]. Один режим протокола предназначен для работы в режиме жесткого реального времени, он использует собственные физический и канальный уровни. Другой режим полностью совместим с Ethernet и позволяет использовать UDP/IP.

Архитектура сети — Master-Slave. Обмен данными организуется мастером и осуществляется циклично. Процесс чтения-записи в ведомых устройствах осуществляется аппаратно, с помощью специального быстродействующего контроллера EtherCAT.

Протокол Powerlink был разработан австрийской компанией Bernecker & Rainer. Протокол Powerlink разделяет сегменты сети на сегмент недетерминированного времени и сегмент реального времени. На физическом уровне в сегменте Real-time используются стандартные IP-пакеты данных, архитектура клиент — сервер, обычные кабельные системы Ethernet. Цикл передачи данных осуществляется по строгому графику в два этапа: критичные по времени доставки данных с типом PDO (Process Data Objects) проходят в первый этап, а трафик с низким приоритетом SDO (Service Data Objects) — второй этап. Данный механизм позволяет избегать коллизий в сети. В рамках одного сегмента обеспечена жесткая детерминированность. Не рекомендуется применение активных устройств в сегменте реального времени. В рамках протокола Powerlink V3 возможна синхронизация нескольких распределенных сегментов по протоколу IEEE 1588. Однако в этом случае детерминированность системы все-таки остается негарантированной. Powerlink гарантирует время цикла передачи данных в пределах 200 мкс с точностью до 1 мкс [23, 24].

ProfiNet является стандартом Ethernet реального времени, совместимым со стеком TCP/IP. Стандарт разработан компанией Siemens

вместо устаревшего Profibus FMS, может работать совместно с Profibus DP (сопряжение через шлюз). Протокол ProfiNet имеет модификации ProfiNet IO для распределенной системы ввода-вывода и ProfiNet CBA для модульных систем управления и объединения сетей ProfiNet (протокол имеет возможность непосредственного подключения к сети устройств полевого уровня) [19]. В ProfiNet существует три режима работы протокола: ProfiNet V1, ProfiNet SRT и ProfiNet IRT. Именно последний режим используется в системах жесткого реального времени. Он гарантирует время цикла в 1 мс с точностью до 1 мкс. В этом случае осуществляется аппаратная реализация. Обмен данными происходит жесткотактируемыми циклами, в которых определена фаза изохронной передачи данных, являющихся критичными ко времени доставки, и фаза передачи данных в формате TCP/IP.

Отметим, что сопряжение уже работающих полевых шин со многими шинами Real-Time Ethernet осуществляется с помощью шлюзов, обеспечивающих бесшовное соединение сетей [19–23].

2.4. Стандартный программный интерфейс контроллеров

Построение ИУС на основе концепции открытых систем предполагает применение снизу доверху программно- и аппаратно-совместимых средств автоматизации различных типов и производителей. Это означает построение системы по модульному принципу с использованием для взаимодействия отдельных модулей только стандартизованных интерфейсов, и обеспечивает возможность ее поэтапной модернизации.

Применительно к уровню контроллеров, аппаратная совместимость означает возможность применения контроллеров разных типов с использованием стандартных системных шин (прежде всего VME и CompactPCI с их клонами), механических конструктивов (IEC 60297) и промышленных сетей (PROFIBUS, MODBUS, CAN и других). Критерий программной совместимости, предполагающий использование мобильного программного обеспечения, которое способно работать на любом контроллере с любой операционной системой в системе управления любой сложности, определил развитие средств програм-

мирования реального времени от традиционного языка ассемблера до CASE-системы ISaGraph и CODESYS.

Язык ассемблера является первым применяемым языком программирования контроллеров. Он обеспечивает получение наивысшей производительности, прямой доступ к оборудованию, возможность вызова любых процедур на других языках. Однако, приложения, написанные на языке ассемблера, не являются переносимыми. Языки ассемблера для различных аппаратных платформ несовместимы, хотя могут быть в целом подобны. В них также отсутствует объектно ориентированный подход.

Первым высокоуровневым языком программирования задач реальном времени является ADA. Язык ADA предоставляет полную среду разработки программ с текстовым редактором, отладочными средствами, системой управлениями библиотеками и т. д. Спецификации ADA 2012 г. закреплены в международном стандарте ISO/IEC 8652. В 1990-х гг. язык ADA был пополнен новыми функциями для объектно ориентированного программирования и программирования в реальном времени. Основным недостатком ADA является его сложность, которая делает язык трудным для изучения и применения. Существующие компиляторы являются дорогостоящими продуктами и требуют мощных процессоров.

Язык C принадлежит к следующему поколению средств программирования систем реального времени. Разработанный для реализации операционной системы UNIX, язык C впоследствии был перенесен на множество других платформ. Конструкции языка C являются очень близкими типичным машинным инструкциям, благодаря чему он нашел применение в проектах, для которых был свойственен язык ассемблера.

Язык C — объектно ориентированный язык программирования C++, высокоэффективный универсальный язык, позволяющий создавать системы любых размеров. Первый стандартизированный на уровне ISO язык программирования — ISO/IEC 14882. Непосредственно в C++ не поддерживается параллельное программирование и программирование в реальном времени, но оно может быть реализовано с помощью специально разработанных программных модулей и библиотек классов.

Язык Java обеспечивает наивысшую переносимость приложения на уровне двоичного кода и является объектно ориентированным язы-

ком. Однако он предоставляет очень низкую эффективность получаемого кода с доступом к оборудованию и вызовами процедур на других языках только посредством библиотечных функций (обычно написанных на C).

К четвертому поколения языков относят CASE-средства (Computer Aided Software Engenering), которые получили широкое распространение при разработке приложений реального времени. Языки четвертого поколения предлагают формализованный способ описания объектов, их свойств и взаимоотношений между собой. По формальному описанию встроенный CASE-компилятор формирует текст приложения на языке C, C++ или Java. Затем этот текст компилируется уже обычным компилятором. Как уже отмечалось ранее, к классу CASE-систем принадлежит графическая среда разработки прикладных программ для программируемых логических контроллеров ISaGRAF.

Среда ISaGRAF является наиболее известной и наиболее широко используемой реализацией стандарта МЭК 61131-3. История пакета ISaGRAF началась с разработки языка программирования контроллеров GRAFCET (кон. 1970-х гг.), который получил статус французского национального стандарта. Впоследствии именно этот язык послужил основой для разработки в рамках МЭК единого стандарта на языки программирования контроллеров.

При разработке стандарта IEC 61131-3 (МЭК 61131-3), который начался в 1979 г. и был утвержден окончательно только в 1992 г., было обнаружено достаточно много вариаций языков контроллеров. Оказалось, что невозможно выбрать только один язык в качестве базового, это привело к созданию нового языка программирования контроллеров, представляющего группу языков с применением современных принципов структурного программирования, абстрактных типов данных, выделения данных и процедур в блок. Графический стиль классических языков для программируемых контроллеров был сохранен. В обоих вариантах языков стандарта введена абстракция управления, и это можно считать главным достижением. Разработчик имеет дело с переменными состояния и не зависящими от типа контроллера способами их обработки, а реальный ввод-вывод вынесен на уровень драйверов (целевых задач). Предложено несколько языков, предназначенных для использования на разных уровнях иерархии системы.

Стандарт IEC 61131-3 [20, 21] описывает следующие языки:

- SFC (Sequential Function Charts) — язык последовательных функциональных схем;

- FBD (Function Block Diagrams) — язык функциональных блоков;
- LD (Ladder Diagrams) — язык релейных диаграмм (релейной логики);
- ST (Structured Text) — язык структурированного текста;
- IL (Instruction List) — язык инструкций.

Язык SFC позволяет формулировать логику программы на основе чередующихся процедурных шагов и переходов, а также описывать последовательно-параллельные задачи в наглядной форме. Он фактически представляет собой средство проектирования прикладного программного обеспечения, которое всегда является комплексом большого числа программных единиц: программ, функциональных блоков, функций. Позволяет (на уровне языка) обеспечить параллельность выполнения программ, установку и контроль состояния порожденных процессов, синхронизацию при приеме и обработке данных, описать состояния автоматизируемого процесса.

Язык SFC представляется наиболее подходящим средством для описания динамических моделей. Для этого описания используется графическая мнемоника, позволяющая наглядно представить логику программы на разных уровнях детализации.

Язык FBD позволяет создать программную единицу практически любой сложности на основе стандартных функциональных блоков (арифметические, тригонометрические, логические блоки, PID-регуляторы, мультиплексоры и т. д.). Программирование сводится к комбинированию и соединению готовых компонентов. В результате этого получается максимально наглядная (используется графическая мнемоника) и хорошо контролируемая программная единица.

Язык LD применяется для описания логических выражений различного уровня сложности и использует в качестве базовых элементов программирования графические элементы релейной логики — «контакты» (contacts) и «обмотки» (coils), связанные с входными и выходными каналами соответственно.

Присутствие в стандарте языка LD определяется, скорее всего, тем, что для релейной техники было разработано огромное количество оборудования и алгоритмов. Он позволяет создавать современные управляющие системы на отлаженной годами алгоритмической базе.

Язык ST относится к классу текстовых языков высокого уровня. По синтаксису он близок к языку Pascal. На его основе можно создавать гибкие процедуры обработки данных. Язык структурированного текста является основным для программирования последовательных

шагов и транзакций языка SFC. Кроме того, его конструкции применимы во всех стандартных языках.

Язык инструкций IL представляет собой унификацию интерфейса языка программирования низкого уровня (ассемблера), не ориентированного на какую-либо микропроцессорную архитектуру. На его основе можно создавать оптимальные по быстродействию программные единицы.

Существует возможность совместного использования программ и процедур, написанных на разных языках, а также возможность вставлять кодовые последовательности на одном языке в коды, написанные на другом. Функциональное расширение возможно за счет поддержки интерфейса языка C.

Пакет ISaGRAF, принадлежащий к классу CASE-систем, включает систему разработки (ISaGRAF WorkBench) и систему исполнения (ISaGRAF Target) [21]. Система разработки предназначена для создания, моделирования, тестирования и документирования прикладных программ, исполняемых под управлением ядра ISaGRAF на системах исполнения, и устанавливается на компьютере под управлением ОС Windows. Система разработки компилирует проект в системно независимый код (Target Independent Code, TIC), загружаемый для исполнения в контроллер. Предусмотрена связь машины разработчика и контроллера по сети. Система исполнения размещается в контроллере и включает в себя ядро ISaGRAF (собственно интерпретатор TIC-кода) и набор модулей связи.

Система разработки поставляется в виде исполняемых модулей, система исполнения для популярных ОС — также в виде исполняемых модулей. Кроме того, поставляется пакет разработки системы исполнения, представляющий собой набор исходных текстов на ANSI C. Этот пакет предусматривает для многозадачной ОС реализацию системы исполнения в виде двух задач:

- ядра целевой задачи;
- задачи связи с системой разработки.

Ядро целевой задачи, называемое еще *виртуальной машиной*, интерпретирует TIC-код и производит опрос модулей ввода-вывода данного контроллера, формирование и вывод на них управляющих воздействий. Ядро работает в синхронном режиме и не использует полностью выделяемый ему временной интервал, позволяя выполняться другим задачам.

Задача связи с системой разработки выполняет следующие функции:

- получает TIC-код приложения и размещает его в разделяемой памяти с ядром целевой задачи;
- получает команды (запросы данных) от среды разработки или от системы SCADA, передает их ядру целевой задачи и ожидает реакции, после чего отсылает ответ.

Взаимодействие этих двух задач также осуществляется через разделяемую память.

Многозадачная архитектура допускает параллельное функционирование нескольких целевых задач и нескольких задач связи в одном контроллере. При этом первая может быть связана с несколькими (до четырех) задачами связи, а вторая — с несколькими (до четырех) целевыми задачами. Это позволяет, в частности, работать через один и тот же порт четырем целевым задачам. Такое разделение функций системы исполнения позволяет ядру работать в режиме жесткого реального времени, т. к. все операции, связанные с непредсказуемыми задержками, вынесены в задачу связи.

Поддерживают стандарт МЭК6-1131/3 программы многих компаний. Этот стандарт в той или иной мере реализует до 80 % предлагаемых на рынке программных продуктов, в т. ч. специализированные системы разработки, предназначенные для конкретных типов контроллеров. В частности, пакет ISaGRAF широко используется для построения прикладных и инструментальных систем.

Общую координацию деятельности производителей и пользователей систем, реализующих стандарт МЭК 6-1131/3, осуществляет независимая Международная ассоциация PLCopen. Она занимается популяризацией и информационной поддержкой стандарта в целях его использования в промышленных системах контроля и управления. Основное направление деятельности ассоциации — поддержка языков программирования, которые позволяют пользователям различных контроллеров обмениваться своими наработками. Одна из важнейших задач PLCopen — разработка системы и принципов сертификации программных продуктов на предмет их соответствия стандарту.

Развитие стандарта МЭК6-1131/3 обеспечило возможность обмена пакетами функциональных блоков между различными платформами и позволило перейти к модульному многоплатформенному программированию.

Еще одно направление развития стандарта МЭК6-1131/3 — переход к распределенным управляющим системам, использующим промышленные сети. Ресурсы ISaGRAF могут обмениваться данными с использованием механизма связывания ресурсов (Resource Binding). Он задает модель взаимодействия как между ресурсами одной целевой машины, так и между ресурсами нескольких целевых машин и определяет принимаемые и обрабатываемые данные, коммуникационные взаимодействия и каналы связи между переменными, каждая из которых описана в рамках своего ресурса.

В процессе развития пакет ISaGRAF перерос в технологию ACP (Automation Collaborative Platform) для обслуживания систем автоматизации. Единая платформа автоматизации разработана как среда, управляемая с помощью открытых подключаемых модулей — плагинов. ACP представляет собой расширяемый слой абстракции с общим интерфейсом, который обеспечивает унифицированные функциональные возможности, выбираемые пользователем. ACP поддерживает одновременно несколько заданных моделей автоматизации (CAM — Concrete Automation Model), предоставляя возможность интеграции разнородных продуктов в единую интегрированную среду разработки.

Среди МЭК-систем программирования выделяется система CODESYS. Комплекс CODESYS предоставляет богатый функционал для разработчика встраиваемых систем: многозадачность реального времени, обработку событий, встроенную визуализацию, развитый набор коммуникаций, быстрое обновление кода, полевые сети, поддержку управления через Интернет, средства национальной локализации проектов и другие функции. CODESYS — продукт немецкой компании 3S-Smart Software Solutions GmbH, специализирующейся на создании универсальных сред программирования на языках МЭК.

CODESYS включает в себя редакторы и трансляторы для всех пяти стандартных языков с рядом существенных расширений. Он также поддерживает значительное число специализированных отладочных и сервисных функций. На сегодня CODESYS — мировой лидер среди МЭК-комплексов. С его помощью ежегодно программируется более полумиллиона контроллеров.

В комплексе CODESYS V3 впервые реализованы оригинальные объектно ориентированные расширения языков МЭК, одобренных третьей редакцией стандарта МЭК6-1131/3 в 2013 г. Таким образом, CODESYS — это широко распространенный высокоэффективный продукт.

2.5. Инструментальные системы диспетчерского управления

Стандарты на программное обеспечение SCADA разрабатываются на уровне предприятия, производства, компании и на отраслевом уровне. Программное обеспечение устанавливается на компьютеры. Для связи с объектом используются драйверы ввода-вывода или OPC/DDE-серверы. Программный код может быть написан на одном из языков программирования или сгенерирован в среде проектирования. Иногда в комплекте SCADA-системы имеется дополнительное программное обеспечение для программирования промышленных контроллеров. Такие SCADA-системы называются интегрированными; к ним добавляют термин SoftLogic [24–27].

Концепция SCADA (рис. 10) обычно предусматривает, что непосредственное управление процессом обеспечивается RTU (УСО — устройство связи с объектом) или PLC (ПЛК — программируемый логический контроллер), а SCADA управляет режимами работы. Например, PLC управляет потоком охлаждающей воды реактора, а SCADA-система может позволить операторам изменять установочные параметры потока воды, менять маршруты движения жидкости, следить за тревожными сообщениями. SCADA-система записывает все нештатные ситуации. Цикл управления с обратной связью проходит через RTU или PLC, в то время как SCADA-система контролирует полное выполнение цикла.

Сбор данных начинается в RTU или на уровне PLC и включает показания измерительного прибора. Далее данные собираются и форматируются таким способом, чтобы оператор, используя HMI (Human-machine interface[1]), мог принять контролирующее решение — корректировать или прервать стандартное управление средствами RTU/PLC [28–29].

Обеспечение многопоточного приема информации, поступающей одновременно от нескольких внешних систем, осуществляется в SCADA в соответствии с одним или несколькими стандартными коммуникационными протоколами, например, IEC 870-5-101, IEC 870-5-104, Modbus, IEEE C37, OPC, ICCP, FDST [25–28].

[1] Человеко-машинный интерфейс.

Для электроэнергетической отрасли, в соответствии со стандартами [27, 28], полный цикл обработки информации, от поступления в SCADA до архивирования и предоставления информации пользователям, должен быть регламентирован и по указанным стандартам не должен превышать 5 с.

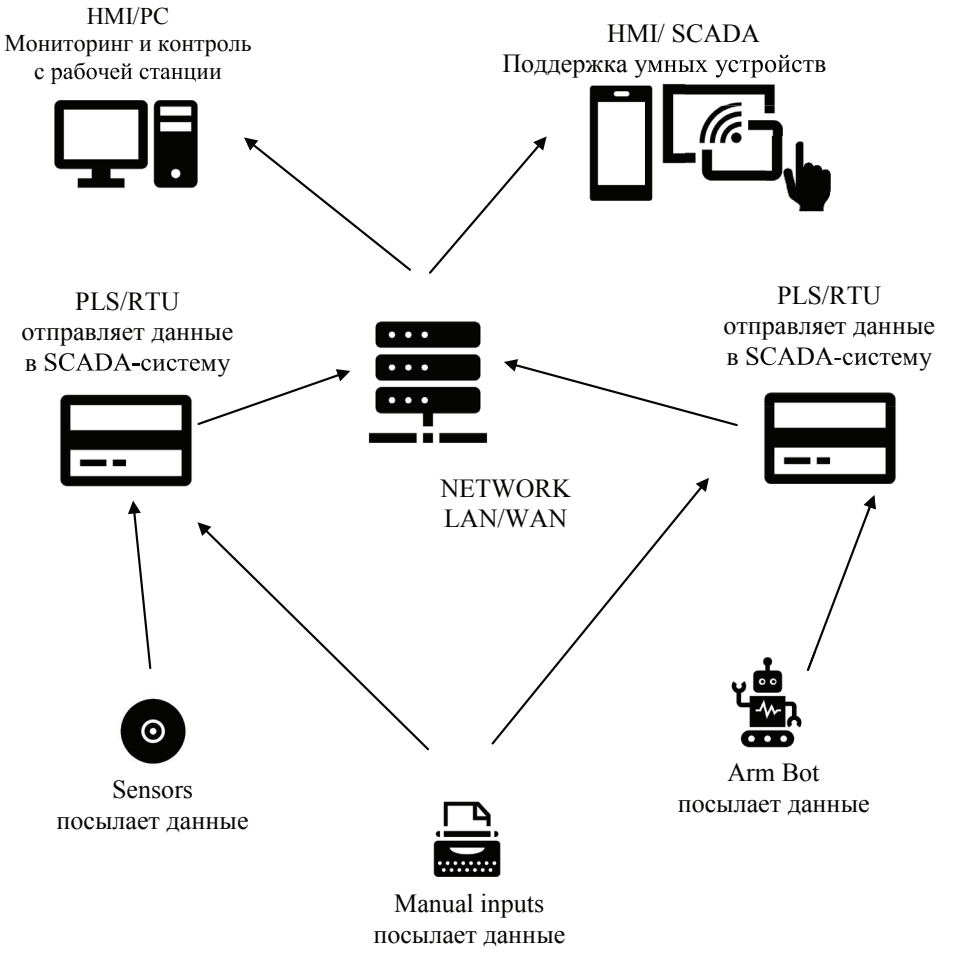

Рис. 10. Базовая архитектура SCADA

SCADA должна поддерживать регламенты обмена данными, обеспечивающие спорадическую (по изменению соответствующего параметра) и периодическую передачу всего объема параметров, в случае запроса данных либо при циклическом опросе.

Устанавливается производительность SCADA как способность обрабатывать не менее определенного числа записей в час (например, 5 млн за один час) и время на поддержку пользовательского интерфейса. В SCADA [28] задержка в выполнении запросов пользователя не должна превышать 5 с, за исключением запросов на большой объем неоднородной информации из различных баз данных.

Требования к номенклатуре и значениям показателей надежности (коэффициенту готовности) регламентируются по ГОСТ Р 27.002–2009.

В России большой популярностью пользуются следующие зарубежные SCADA:

- WinCC (Siemens, Германия);
- InTouch (Wonderware, США);
- RSView32 (Rockwell Automation, США);
- Genesis64 (Iconics, США);
- Vijeo Citect (Schneider Electric, Франция).

Среди отечественных SCADA-систем хорошо зарекомендовали себя:

- MasterSCADA (ИнСАТ, Москва);
- TRACE MODE (AdAstra, Москва);
- Круг 2000 («Круг,» Пенза).

В отличие от большинства западных SCADA, все российские системы содержат встроенные средства программирования контроллеров с использованием языков стандарта МЭК6-1131/3, включая язык функциональных блоков. Исполнительная система для контроллеров SCADA-систем, рассчитанных на работу в среде Windows на PC-совместимых компьютерах, может работать на платформе Logix и некоторых других платформах.

Стандарт OPC (Open Platform Communications[1]) поддерживает все перечисленные системы, однако в системе «Trace Mode» упор делается на использование собственных драйверов, а MasterSCADA, хоть и поддерживает использование драйверов, основывается на технологии «OPC в ядре системы» и предлагает отдельный инструментальный пакет для разработки OPC-серверов.

В табл. 4 приведен список некоторых свободно распространяемых систем SCADA и их характеристики.

[1] В задачи OPC входит предоставление разработчикам промышленных программ универсального фиксированного интерфейса (т. е. набор функций) обмена данными с любыми устройствами. В то же время разработчики устройств должны предоставлять программу, реализующую этот интерфейс (набор функций).

Таблица 4

Свободно распространяемые SCADA-системы (Free SCADA Software)

Наименование SCADA	Описание	Характеристики	Сайт
Mango Automation Mango Automation 3.7.7 Free	Гибкое программное приложение, которое позволяет просматривать, регистрировать, строить графики, мнемосхемы, составлять отчеты по данным с датчиков, оборудования, ПЛК, баз данных, веб-страниц и т. д.	Кросс-платформенная (Windows, Linux, or Mac). База данных — NoSQL. Отчеты с автоматическими уведомлениями по электронной почте. Интернет — 100 % веб-совместимая	URL: https://store.infiniteautomation.com/core
PROMOTIC Visualization software (*рус*. ПО визуализации)	Набор инструментов объектного ПО SCADA для создания приложений, которые выполняют функции наблюдения, контроля и отображения процесса в различных промышленных областях	Поддерживает ОС: Windows 10/8/7/Vista/XP/Embedded/2003-12Server and higher. База данных — SQL- и ODBC-интерфейс для базы данных. Веб — Internet/Intranet Interface: XML, OPC, ActiveX, DDE	URL: https://www.promotic.eu/en/promotic/download/download.htm
WINLOG Lite	Бесплатная версия программного обеспечения SCADA Winlog Pro, которая позволяет разрабатывать и выполнять веб-приложения, доступна через браузер, имеет рабочий цикл 60 мин, по истечении которого останавливается взаимодействие с датчиками.	Поддерживает ОС: Windows 10 (32/64 bit), Windows 8.1 (32/64 bit), Windows 8 (32/64 bit), Windows 7 (32/64 bit), Windows Embedded 7, Windows Server 2012, Windows Server 2008 R2	URL: https://www.sielcosistemi.com/en/download/public/download.html

Продолжение табл. 4

Наименование SCADA	Описание	Характеристики	Сайт
IGSS FREE50	ISGSS, используемый для мониторинга и управления производственными процессами, представляет собой масштабируемое решение для взаимодействия со всеми основными отраслями промышленности PLC	Поддерживает: 32-битную & 64-битную версии: Windows 2016 Server, Windows 10 (только IGSS 11), Windows 8.1 (только IGSS 10), Windows 2012 Datacenter (для всех пакетов сервиса), Windows 2012 Essentials (для всех пакетов сервиса) и т. д. Базы данных: Microsoft SQL Server 2005, Microsoft SQL Server 2008, Microsoft SQL Server 2008 R2, Microsoft SQL Server 2012 Отчеты: Microsoft Office 2007, Microsoft Office 2010, Microsoft Office 2013 Идентифицирует вплоть до 50 объектов в рамках проекта. Собирает данные от ПЛК без каких-либо ограничений по времени. Использует один из +70 PLC-драйверов. Поддержка драйверов ПЛК: http://igss.schneider-electric.com/products/igss/product-information/plc-drivers-supported.aspx	URL: http://igss.schneider-electric.com/products/igss/download/free-scada.aspx

Продолжение табл. 4

Наименование SCADA	Описание	Характеристики	Сайт
INTEGRAXOR	Система на базе ОС Windows предоставляет IDE (интегрированную среду разработки) для разработки, настройки и управления HMI (человеко-машинный интерфейс) и приложениями SCADA	Современная и многообещающая Ecava IGX SCADA разработана с нуля. Предназначена для простой и быстрой передачи данных через интернет. Использует открытые технологии в строгом соответствии с передовыми практиками. Запускается на всех платформах Windows от Windows XP до Windows 10. ПО совместимо, просто в использовании и эффективно. Может работать как встроенное приложение Windows. Масштабируемо во всем диапазоне Windows Server, от 2008 до Windows Server 2012	URL: https://www.integraxor.com/download-scada/
MYSCADA Бесплатная версия для некоммерческого использования	Большинство инструментов SCADA работает в Linux и поддерживает визуализацию SMART. Очень подходит для использования в различных отраслях промышленности, а именно: в нефтегазовой отрасли, автоматизации зданий, водоснабжении и очистке сточных вод, энергетике и на производственных линиях	Поддерживает: Windows 32/64 Bit, Max OSX, *Linux* 32/64Bit, Android OS, iOS	URL: https://www.myscada.org/download/

Окончание табл. 4

Наименование SCADA	Описание	Характеристики	Сайт
FREESCADA	Основана на дизайне с открытым исходным кодом для ОС Windows. Предоставляет конечным пользователям большие возможности для визуализации и управления производственными процессами	Использует OPC-сервер для сбора данных. Разработана на языке Си# на платформе Dot Net 3.0	URL: https://sourceforge.net/projects/free-scada/

Дальнейшее принципиальное развитие SCADA-систем ожидается по двум направлениям. Одно из направлений — это развитие облачного сервиса для решения проблемы хранения большого числа архивируемых данных и доступа к ним различных групп пользователей. Например, в MasterSCADA 4D реализованы системные узлы «сервер» и «облачный сервер», что позволяет снизить сетевые накладные расходы, организовать получение данных между узлами в форме групповых запросов, интегрировать в единый проект множество удаленных предприятий.

Другое направление связано с диверсификацией пользовательских интерфейсов (multiexperience), что к 2028 г. должно изменить способы взаимодействия людей с системами автоматизации. Уже в настоящее время платформы разговорного искусственного интеллекта, решения в области дополненной, виртуальной и смешанной реальности, сенсорные технологии демонстрируют новые возможности такого взаимодействия. В ближайшие годы прогнозируется все более широкое проникновение этих возможностей в рабочие процессы. И главное ожидаемое изменение, по мнению аналитиков, состоит не в разнообразии и удобстве интерфейсов, а в том, что технологии будущего станут предугадывать наиболее подходящие для пользователя способы взаимодействия [29].

Виртуализация производства является ключом к трансформации текущего производства в новое современное предприятие [30]. Процесс начинается непосредственно с уровня цеха: виртуализируются его функции, реализуются соответствующие имитации и рассчитывается динамика производства. Благодаря цифровым моделям определяется наиболее эффективная модель работы производства. Таким образом, средствами компьютерного моделирования можно создать заводской цех, управлять его параметрами, находить узкие места, выявлять причины затрат, уменьшать выбросы углекислого газа. Эта технология масштабируется в широких пределах и применима даже к одному конкретному продукту, позволяя смоделировать его поведение в реальном производстве в то время, пока он еще находится на чертежной доске [29–31].

Еще одним новым направлением в области автоматизации управления производством является OMI[1] — интерфейс управления операциями в промышленности. Для реализации всех нужд современного оператора, управляющего процессами, уже недостаточно традиционных подходов и систем, таких как HMI, SCADA. Это связано с высоким уровнем автоматизации на предприятиях, которая охватывает самые разные сферы контроля управляемого процесса. Это и данные с видеокамер, установленных рядом с оборудованием, и мессенджер, предоставляющий общение с коллегой, и системы ERP/MES/MOM, а также компоненты и приложения для глубокого анализа аварийных данных. Статистика показывает, что современному оператору крупного диспетчерского центра требуется работать с 10–20 дополнительными приложениями, помимо SCADA. OMI позволяет решить этот вопрос за счет интеграции внутрь интерфейса дополнительных приложений. Получая уведомление об аварийном состоянии оборудования, OMI автоматически подсвечивает для оператора всю необходимую информацию для наискорейшего устранения аварии. Нажав на мнемосхему, оператор видит видеопоток именно с нужной камеры, анализ предыдущего состояния именно по этому оборудованию, регламентную документацию именно по этому событию, а в мессенджере подсвечивается контакт персоны, ответственной за ремонт именно этого оборудования.

В России уже есть реализованные на данной платформе диспетчерские или операционные центры — в крупнейших компаниях в об-

[1] Operation Managemant Interface.

ласти нефте- и газодобычи и в горной промышленности [30, 31]. Эти решения базируются на концепции Industrial Internet of Things (IIoT).

Промышленный интернет вещей — это многоуровневая система, включающая в себя датчики и контроллеры, установленные на узлах и агрегатах промышленного объекта, средства передачи собираемых данных и их визуализации, мощные аналитические инструменты интерпретации получаемой информации и многие другие компоненты. Промышленный интернет вещей фактически является новой бизнес-моделью управления в эпоху индустрии 4.0 [32, 33].

Концепция развивается на базе технологии интернета вещей (IoT — Internet of things). Технология предлагает подключение к вычислительной сети физических предметов («вещей»), оснащенных встроенными технологиями их автоматической идентификации, например радиочастотными или оптическими идентификаторами (штрихкоды, Data Matrix, QR-коды, радиочастотные метки — рис. 11). Активному внедрению IoT способствует широкое распространение беспроводных сетей и облачных вычислений. Такие технологии широко применяются в логистике и на транспорте.

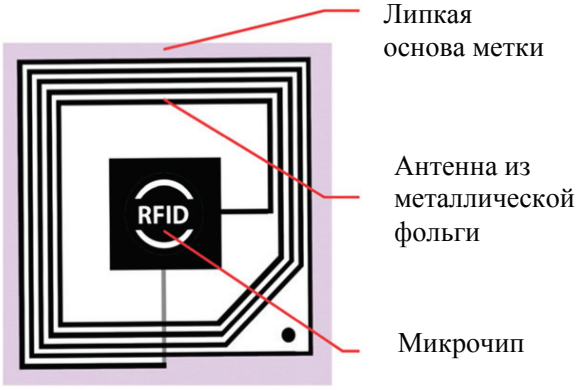

Рис. 11. Структура радиочастотной метки

В интернете вещей используется широкий класс средств измерения, от элементарных датчиков и приборов учета потребления (таких как интеллектуальные счетчики) до сложных интегрированных измерительных систем. Сетевые технологии обычно базируются на стандарте IEEE 802.15.4, который определяет физический уровень и управле-

ние доступом для персональных сетей с низким уровнем мощности. Стандарт составляет основу таких протоколов, как ZigBee, WirelessHart, MiWi, 6LoWPAN, LPWAN.

Технологический эффект IIoT, отмечаемый менеджментом компаний на протяжении уже четырех лет, — это изменение технологий управления, а не технологий производства. Именно стек технологий управления и автоматизации управления, в отличие от предыдущих технологических (промышленных) революций, определяет переход к новому технологическому укладу — четвертой промышленной революции.

По данным IDC, общий мировой объем капиталовложений в направления, связанные с интернетом вещей, в 2016 г. составил 737 млрд долл., в 2017 — более 800 млрд долл.; к 2021 году прогнозируются инвестиции порядка 1,4 трлн долл. [35].

Стандартизированные на национальном уровне протоколы интернета вещей (IoT) NB-Fi, LoRaWAN RU и OpenUNB будут включены в проект международного стандарта совместимости систем IoT/IIoT. Такое решение было принято в ноябре 2019 г. на заседании подкомитета Международной организации по стандартизации (International Organization for Standardization, ISO) и Международной электротехнической комиссии (International Electrotechnical Commission, IEC) в области интернета вещей, прошедшем в Санкт-Петербурге. В России разрабатывается собственная технология для сетей LPWAN (узкополосные беспроводные сети связи интернета вещей), которая может конкурировать с существующими в мире технологиями и протоколами [28, 34, 35].

Масштабный переход на SCADA с поддержкой IIoT не означает полного отхода от SCADA третьего поколения. Модернизация существующих SCADA-систем достаточно сложный и длительный процесс, и многие отрасли промышленности продолжают полагаться на существующие решения, например MasterSCADA 3.X. Реализация IIoT — следующий этап в эволюции SCADA, который обеспечит высокую эффективность при небольших затратах на разработку и внедрение. SCADA-система четвертого поколения позволяет получать и хранить большие объемы данных со всех устройств, предоставляя к ним контролируемый доступ. Поддержка OPC UA, облачных сервисов, HTML5 и защита от киберугроз составляют неотъемлемую часть функций современных SCADA-систем [31].

SCADA-системы являются очень уязвимыми с точки зрения безопасности. В 2010 г. с использованием вируса Stuxnet была осуществлена атака на центрифуги для обогащения урана в Иране. Компьютерная группа реагирования на чрезвычайные ситуации (ICS-CERT) 26 мая 2016 г. предупредила об уязвимостях SCADA-систем с использованием web-технологий, позволяющих удаленно вносить конфигурационные изменения и управлять процессами.

Обеспечение безопасности промышленных систем управления (ICS[1]), объединяющих не только SCADA, но и распределенные системы управления (DCS[2]), подразумевает в первую очередь их защиту от любых преднамеренных или непреднамеренных помех, воздействие которых может привести к нарушениям функционирования системы. Основополагающим стандартом в этом вопросе является международный стандарт IEC 61511:2016 «Functional safety — Safety instrumented systems for the process industry sector»[3], разработанный для индустриального сектора. Однако этот стандарт еще требует оценки приборной системы безопасности (Safety Instrumented System, SIS) в системах управления [29, 32].

Правительства и международные отраслевые организации разработали и ввели в действие ряд стандартов, позволяющих повысить устойчивость систем к воздействию потенциальных угроз. Приведем некоторые из основных стандартов подобного типа:

- серия стандартов ISA99 — Industrial Automation and Control Systems Security («Безопасность промышленной автоматизации и систем управления»);
- международный стандарт IEC 62443;
- стандарт Национального института технологий стандартов США (National Institute for Standards Technology, NIST) SP 800-82 — Guide to Industrial Control Systems Security standard («Руководство по безопасности промышленных систем управления»);
- серия стандартов безопасности CIP Совета североамериканских штатов по надежному обеспечению электроэнергией (North American Electric Reliability Council, NERC).

Функциональная безопасность, включая кибербезопасность, имеет свой жизненный цикл, который представляет непрерыв-

[1] Industrial Control System.

[2] Distributed Control System.

[3] «Функциональная безопасность — системы безопасности приборные для промышленных процессов».

ный процесс функционирования системы безопасности и зависит от трех основных компонентов: анализа, реализации и обслуживания [29, 32].

Возникновение новых киберугроз требует, чтобы управление рисками безопасности также было непрерывным процессом, проводился постоянный аудит и проверка всего жизненного цикла кибербезопасности. Это включает управление внесением исправлений (патч-менеджмент), обновление антивирусных программ и баз, осведомленность о тенденциях и рисках в области безопасности для конкретной отрасли.

Контрольные вопросы для самопроверки

1. Укажите суть и значение принципа неразрабатываемых заново приложений. Кем был выдвинут этот принцип и какую роль сыграл он в развитии компьютерных технологий?
2. Что такое COTs, MOTs и ROTs?
3. Дайте определение открытым системам. Что такое POSIX?
4. Какая модель разрабатывалась для стандартизации сетевых технологий? Почему кроме модели OSI/ISO на практике применяются и другие?
5. Что такое международный стандарт «Евромеханика»? Опишите набор требований конструктива. Какие задачи решает конструктив?
6. Что такое крейт? Как он устроен? Как устроена рабочая стойка?
7. Какие задачи решает открытый компьютерный проект?
8. Приведите примеры интерфейсов встраиваемых систем 1970–1990-х гг. и их отличие от с современных интерфейсов.
9. Сравните стандартизованные интерфейсы VME, Compacn PCI и Fastbus.
10. Что позволило интерфейсу VME, созданному в 1980-х гг., оставаться востребованным и в наши дни?
11. Определите основные недостатки интерфейса Compact PCI. В чем состоят главные изменения (включая нововведения, исключения), внесенные в Compact PCI при переходе к PCI Express, COM Express и с PCI Serial?
12. Для каких целей разрабатывался стандарт Fastbus?

13. Сравните процесс передачи данных в стандартизованных интерфейсах VME, Compct PCI и Fastbus.

14. Сравните механизмы арбитража шины в стандартизованных интерфейсах VME, Compct PCI и Fastbus.

15. Какие основные направления развития магистрально-модульных систем можно констатировать в настоящее время?

16. Что называют промышленной сетью? Какие задачи решают промышленные сети? Назовите преимущества и недостатки промышленных сетей.

17. Какие коммуникационные технологии входят в международный стандарт IEC 61158?

18. Сравните топологии трех технологий стандарта IEC 61158: P-Net, CAN, ProfiBus.

19. Сравните сетевые взаимодействия, реализуемые в рамках технологий стандарта IEC 61158: P-Net, CAN, ProfiBus.

20. Сравните механизмы арбитража, реализуемые в рамках технологий стандарта IEC 61158: P-Net, CAN, ProfiBus.

21. Что понимают под развитием инфраструктуры промышленных сетей?

22. Опишите стандарт Real-Time Ethernet. Какие преимущества исходного стандарта Ethernet были использованы для развития технологии реального оборудования для реального времени?

23. Укажите особенности объектно ориентированного протокола CIP для Ethernet/IP.

24. Укажите преимущества и недостатки протокола EtherCAT.

25. Какие режимы работы существуют в стандарте ProfiNet?

26. Почему является востребованной стандартизация средств программирования ИУС?

27. Какие языки программирования используют непосредственно команды процессора контроллера? Какие языки программирования являются графическими (проблемно ориентированными)?

28. Назовите языки программирования в реальном времени первого, второго, третьего и четвертого поколений. Укажите, какими новыми функциями был пополнен каждый язык программирования в сравнении с языком предыдущего этапа.

29. Что такое класс CASE-систем и какие языки принадлежат этому классу?

30. Что определяет стандарт МЭК 1131-3? Как удалось реализовать в указанном стандарте современные принципы структурного программирования, абстрактных типов данных, выделения данных и процедур в блок?

31. В чем состоит особенность языка программирования контроллеров IsaGRAF? Укажите систему разработки и систему исполнения пакета IsaGRAF. Как они функционируют для многозадачной ОС?

32. Что такое Единая платформа автоматизации? Какие задачи она реализует?

33. Укажите особенности пакета ISaGRAF PRO.

34. Что такое комплекс CODESYS? Какой функционал он предоставляет разработчику?

35. Сравните компилятор CODESYS с компилятором C++ по трансляции кода, среде программирования и мобильности.

36. Сравните компилятор CODESYS с компилятором C++ по разделению между верхним, прикладным, пользовательским уровнем управляющей программы и системным уровнем.

37. В чем состоит концепция SCADA? Представьте базовую архитектуру SCADA. Приведите примеры наиболее известных SCADA-систем в России и мире.

38. Особенности стандарта OPC.

39. Приведите пример свободно распространяемых SCADA-систем и охарактеризуйте их.

40. Сравните SCADA-системы InTouch и Trace Mode по характеристикам, техническим возможностям и укажите направления их дальнейшего развития.

Для промышленных ИУС выделяют специализированную разновидность IoT — промышленный интернет вещей (IIoT — Industrial IoT). По сравнению с обычными системами IoT, IIoT должен обеспечить:

- высокий уровень безопасности;
- реализацию всех уровней модели ИУС, включая верхние;
- большое количество источников и получателей данных («вещей»);
- большие объемы передаваемой информации.

По сравнению с классическим построением сетевой инфраструктуры ИУС на основе соединяемых через шлюзы изолированных фрагментов, IIoT на основе единой сети должен обеспечивать большие объемы данных, передаваемые на верхние уровни.

При сохранении иерархической структуры ИУС (см. рис. 1) шлюзы между уровнями этой структуры должны, с одной стороны, обеспечить передачу больших объемов информации, с другой — отфильтровать недопустимые по политике безопасности запросы, изолируя тем самым сети разных уровней.

Построение единой АС предприятия, объединяющей отдельные АС и ИУС, позволяет обеспечить высокое качество управления. При принятии управленческих решений, в такой системе обрабатываются большие массивы информации с использованием методов и алгоритмов обработки больших данных и искусственного интеллекта. Автоматизированная система такого уровня образует киберфизическую систему, в которой возможно полное исключение человека из производственных и бизнес-процессов. При построении киберфизической системы, чаще всего используют возможности вычислений и обработки данных в централизованных или распределенных центрах обработки данных (ЦОД) произвольной, переменной и динамически изменяемой структуры. Такие технологии традиционно называют «облачными» (Cloud).

«Промышленное облако» (IC — Industrial Cloud) отличается следующими характеристиками: организация взаимодействия промышленных устройств; безопасность обмена конфиденциальными сведениями между всеми участниками производственной цепочки.

При организации IIoT выделяют несколько слоев: аппаратный, сетевой, сервисный и контентный. Аппаратный слой включает производственное оборудование и различные датчики. Сетевой слой — протоколы коммуникации и взаимодействие с публичными облаками. Сервисный слой содержит основные программы, в которых осуществляется интеллектуальная аналитика данных. Контентный слой необходим для предоставления пользователю доступа к результатам работы сервисного слоя и принятия управленческих решений.

Облачные и туманные вычисления относятся к активно развивающимся технологиям последнего десятилетий. Они встраиваются в процессы автоматизированного управления технологическими процессами, производством и бизнесом в целом. На рис. 12 представлена иллюстрация логических уровней ИУС с облачными сервисами и туманной сетью. Облачные технологии создают возможность хранить файлы и пользоваться программами, что позволяет экономить ресурсы на создание локальных программ. На рис. 12 облачные сервисы представлены самым верхним уровнем.

Модель облачных вычислений состоит из внешней и внутренней части. Эти два элемента соединены по сети через интернет. Внутренняя часть предоставляет приложения, компьютеры, серверы и хранилища данных — сервисов, создающих облако. Внешняя часть состоит из клиентского компьютера или сети компьютеров предприятия и приложений, используемых для доступа к облаку. Посредством внешней части пользователь взаимодействует с системой.

Серверы в облачной технологии работают в едином информационном поле. Для распределения ресурсов создаются виртуальные серверы. На одном физическом сервере может быть организовано несколько виртуальных серверов [37].

Рассматривают несколько моделей развертывания облака: частное облако, публичное облако и гибридное облако. Облака могут специализироваться на программном обеспечении (сервисная модель — SaaS), разработке и предоставлению платформ (сервис PaaS), предоставлении инфраструктуры (сервис IaaS) и др. В России разработка облачных сервисов осуществляется в соответствии со стандартом ISO/IEC 17788–2016 «Информационные технологии — облачные вычисления».

Многие известные компании, такие как Microsoft, Google, Amazon, «Яндекс», организовывают облачную инфраструктуру для упрощения

работы пользователей, а также для уменьшения нагрузки на локальные устройства.

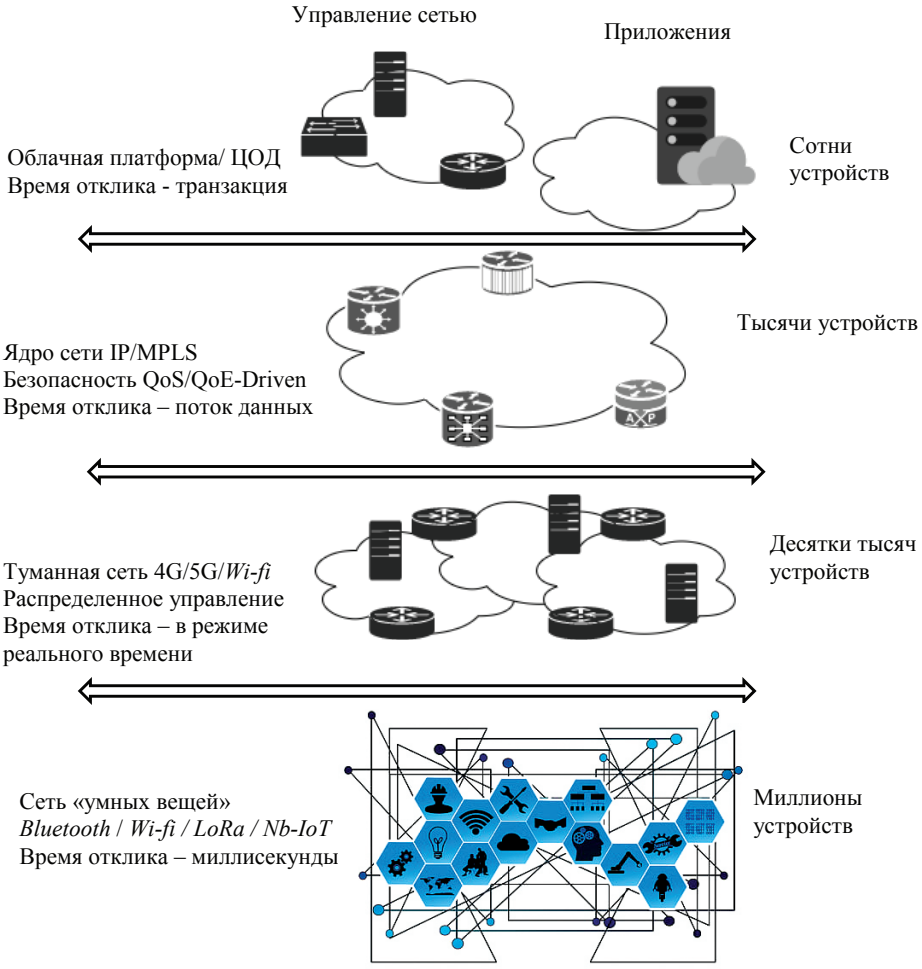

Рис. 12. Логические уровни ИУС
с облачными сервисами и туманной сетью

Отметим концепцию граничных вычислений Edge computing как промежуточный подход, который дает возможность оптимизировать облачную архитектуру путем переноса части алгоритмов обработки данных ближе к их источнику — объекту управления. Подход позволил использовать преимущества облака: неограниченную вычислительную мощность, плату только за потребленные ресурсы, простоту

создания распределенных решений и большое число готовых сервисов. Кроме того, в концепции граничных вычислений устранены основные недостатки облачных вычислений: сложности с работой системы при отсутствии связи, невозможность передавать данные по сети из-за их большого объема или секретности этих данных. В соответствии с этой концепцией, на граничные устройства возлагаются задачи приема и предварительной обработки первичных данных от объекта мониторинга по промышленным протоколам, оптимизации объемов передаваемых в облако данных с их предварительной обработкой, а также оперативной обратной связи и надежного исполнения команд из управляющего центра (облака) с минимально возможным временем отклика.

Концепция туманных вычислений [38] предполагает дополнительный уровень работы с информацией как в локальной, так и в глобальной сети, занимая промежуточное положение между облачными дата-центрами, конечными устройствами и другими элементами инфраструктуры данных. Туманные вычисления представляют еще один уровень сбора и анализа данных, более близкий к пользователю, в то время как граничные вычисления (Edge computing) являются ближайшей к конечным устройствам точкой описываемой сети (см. рис. 12).

Выделим важные преимущества современных ИУС, которые ей придает технология туманных вычислений:

- более широкую географию сетей;
- мобильность;
- очень большое количество узлов внутри сети данного типа;
- улучшенные возможности использования технологий беспроводного доступа;
- расширенные возможности для работы потокового программного обеспечения и приложений, работающих в реальном времени;
- неоднородность вычислительных сетей.

Процесс внедрения стандартов в сфере облачных и туманных вычислений является мощной движущей силой в развитии современных ИУС. Он делает данные более доступными, позволяет эффективнее взаимодействовать программному обеспечению и платформам, предлагает стандартные протоколы, которые легче использовать. Указанные тенденции приводят, с одной стороны, к снижению стоимости реализации, эксплуатации и обслу-

живания облачных и туманных технологий. С другой стороны, наблюдается лавинообразный рост использования данных технологий. Ниже представлены организации, развивающие открытые стандарты на технологии и компоненты облачных сервисов.

Технологии, компоненты и стандарты по облачным сервисам, продвигаемые различными международными и национальными организациями:

IBM	развитие облачной операционной системы с открытым исходным кодом OpenStack; управление облаком — IBM Smart Cloud Orchestrator и IBM Cloud Manager с OpenStack; технология Cloud Foundry, которая помогает разработчикам быстро создавать веб- и мобильные приложения с интеграцией нескольких языков программирования, структур и служб по мере необходимости; программно определяемые сети предоставляют уровень абстрагирования API для управления сетью; технология автоматизированного развертывания ПО с открытым исходным кодом (инструменты — Puppet, Juju или Chef); технология безопасности OAuth для интеграции REST API-интерфейсов на предприятии.
Distributed Management Task Force (DMTF)	стандарт OVF (Open Virtualization Format), принятый как ISO 17203, который обеспечивает единый формат программного обеспечения, ориентированного на виртуальные системы.
Cloud Management Working Group (CMWG)	использование и развитие технологии Cloud Infrastructure Management Interface (CIMI) для визуального представления всего жизненного цикла облачной службы, так что можно улучшить реализацию и управление этой службой и гарантировать ее соответствие предъявляемым требованиям.
Cloud Auditing Data Federation Working Group (CADF)	стандарт DMTF CADF — модель, которую программисты, менеджеры и пользователи применяют для самостоятельного аудита безопасности приложений.
ETSI Technical Committee Cloud	стандарты в сфере мобильных облачных вычислений, охватывающих масштабируемость, передачу данных и безопасность.

Global Inter-Cloud Technology Forum (GICTF)	Стандартизация сетевых протоколов и интерфейсов в целях создания более надежной сети облачных услуг, которая решает проблемы безопасности, качества данных, системы реагирования и надежности.
Международная организация по стандартизации (ISO) либо Международная электротехническая комиссия (IEC)	создание среды разработки JTC 1, с помощью которой формируются международные стандарты для корпоративных и потребительских приложений. Работая с JTC 1, технические специалисты создают основополагающие технологические инфраструктуры и интегрируют сложные технологии. Стандарты, выпущенные этой группой за последние годы: • Cloud Computing Service Level Agreements (CCSLA); • Cloud Computing Interoperability and Portability (CCIP); • Cloud Computing Data and its Flow (CCDF); • Cloud Data Management Interface (CDMI).
Object Management Group (OMG)	унифицированный язык моделирования OMG (Unified Modeling Language™ — UML), используемый как основа для моделирования структуры приложений, структуры данных, бизнес-процессов и архитектуры. Его совместное использование с UML Meta Object Facility (MOF™) и Model-Driven Architecture® позволило унифицировать весь процесс разработки, что помогает частично решить проблемы переносимости, взаимодействия и многократного использования облачных ресурсов.
Open Grid Forum (OGF) либо Open Cloud Computing Interface (OCCI)	спецификация OCCI «RESTful-протокол и API-интерфейс для решения задач управления всех типов» распространяется через OGF, и предлагает различные реализации и инструменты общего назначения, и ориентирована на интеграцию, переносимость, взаимодействие, а также автономное масштабирование и мониторинг.

В России активно используется облачный сервис OwenCloud (ОВЕН). Сервис предоставляет возможность подключать и контролировать приборы к SCADA-системам через OPC-серверы [38–40]. Сервис OwenCloud обеспечивает высокий уровень надежности и безопасности. Данные передаются в зашифрованном виде и хранятся на защищенных серверах в ЦОД, не могут быть изменены или искажены

атакой типа MITM. Интерфейс облачного сервиса OwenCloud позволяет отслеживать и изменять параметры приборов в режиме реального времени Действия пользователей из сервиса при необходимости могут быть ограничены. Для особо важных действий, которые могут навредить работе приборов, можно настроить их подтверждение через коды, рассылаемые по SMS. Подобный механизм используется банками для подтверждения операций, совершаемых чрез онлайн-сервисы.

Серверы OwenCloud обеспечивают высокий уровень отказоустойчивости и резервирования, снижающий почти до нуля вероятность потери данных.

Базовые функции облачного сервиса для приборов ОВЕН предоставляются бесплатно. Существует ряд ограничений на время хранения данных, количество рассылаемых SMS и т. д. На платной основе могут предоставляться дополнительные возможности по обработке и визуализации данных.

Контрольные вопросы для самопроверки

1. Что такое виртуализация производства? Приведите примеры использования данной технологии.
2. Что такое интернет вещей? Почему промышленный интернет вещей рассматривают как новую бизнес-модель управления в эпоху индустрии 4.0.
3. Как решается проблема безопасности промышленных систем, включающих SCADA? Какие стандарты обеспечивают повышение устойчивости систем к воздействию потенциальных угроз?
4. Что означают облачные и туманные вычисления?
5. Опишите архитектуру современной ИУС с использованием облачных сервисов. Приведите примеры современных облачных сервисов.
6. Какие преимущества предоставляют облачные сервисы?
7. Какие преимущества привносят туманные вычисления в современные ИУС?
8. Какие технологии и стандарты продвигают российские и зарубежные компании в области облачных сервисов?
9. Какие технологии и стандарты продвигает международная организация стандартов в области облачных сервисов?
10. Приведите пример российского облачного сервиса.

4. Экономическая эффективность ИУС

Введение концепции открытых систем позволило заложить прежде всего экономическую базу для развития возможностей современных ИУС и подходов их построения, что повлекло широкое распространение аппаратных и программных продуктов по всему миру, их бурное развитие и удешевление.

Цифровая трансформация невозможна без отлаженной системы управления бизнес-процессами — в равных конкурентных условиях побеждает организация с наиболее эффективными процессами. В связи с этим современные проблемы ИУС решаются в плоскости управления внутренними ресурсами и внешними связями, эффективности роботизации бизнес- процессов, и поэтому уровень современной ИУС оценивается по уровню зрелости системы управления производственными процессами и бизнес-процессами [39, 40].

Модели и стандарты управления производством и производственной деятельностью, сформировавшиеся еще в 1980–1990-х гг., постоянно совершенствуются с использованием специализированного прикладного программного обеспечения MES [41, 42]. Этими моделями дополняют следующие стандарты:

- стандарт ISA-95 «Интеграция систем управления предприятием и технологическим процессом» («Enterprise-Control System Integration») определяет единый интерфейс взаимодействия уровней управления производством и компанией с рабочими процессами и производственной деятельностью отдельного предприятия [43];
- стандарт ISA-88 «Управление периодическим производством» («Batch Control») определяет технологии управления периодическим производством, иерархию рецептур, производственные данные [43];
- модель процессов цепочки поставок (Supply-Chain Operations Reference, SCOR) представляет референтную модель управления процессами множественных поставок, связывающую деятельность поставщика и заказчика. Модель SCOR описывает бизнес-процессы для всех фаз выполнения требований

заказчика. Раздел SCOR «Изготовление» («Make») посвящен производству [43, 44].

Организационная стратегия интеграции производства и операций управления трудовыми ресурсами, финансового менеджмента и управления активами, ориентированная на непрерывную оптимизацию ресурсов предприятия, реализована в ERP-системе (Enterprise Resource Planning — планирование ресурсов предприятия), которая представляет интегрированный пакет прикладного программного обеспечения.

Развитием ERP-системы стал стандарт ERP II, определяющий корпоративную информационную систему, открытую для всех участников в поле общих интересов бизнеса. Стандарт ERP II дополнил ERP-системы функциональностью CRM[1], SCM[2] и электронной коммерции.

В системах класса ERP II коренным образом изменилась стратегия работы. Система начала работать не только в рамках одной организации: результаты внутренних процессов организации видны и для контрагентов. Система стала интегрированной в информационное пространство за пределами предприятия, что позволяет вовлекать контрагентов в более тесное сотрудничество (рис. 13).

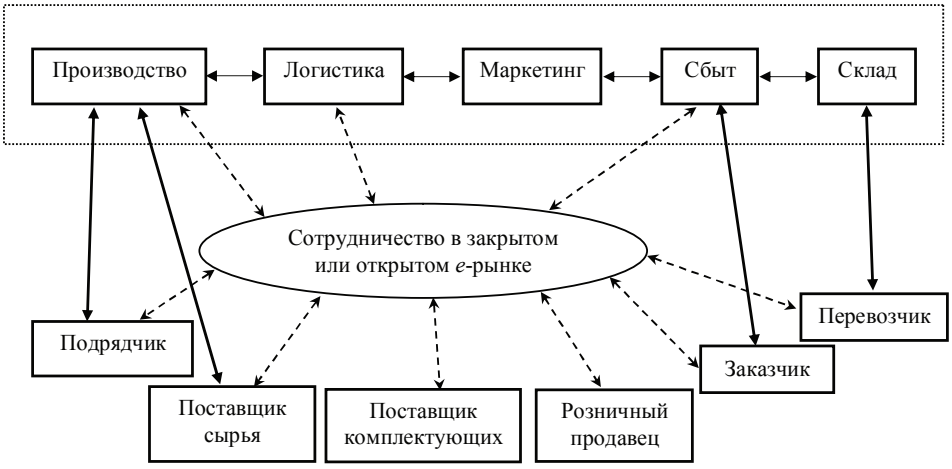

Рис. 13. Процессы интеграции ERP II-систем

Кроме Enterprise Recourse Planning (ERP) — систем планирования ресурсов предприятия, выделяют еще один класс кор-

[1] Customer Relationships Management — управление взаимоотношениями с клиентами.

[2] Supply Chain Management — управление цепочками поставок.

поративных информационных систем — Business Management Systems (BMS)[1]. К ним относятся системы класса «Low End PC», рассчитанные на одно рабочее место или небольшие сети из 4–8 компьютеров, системы класса Middle PC, отличающиеся большей глубиной и широтой охвата функций для описания десятков бизнес-процессов, и системы High End PC, рассчитанные на работу на средних предприятиях большого числа пользователей, без предъявления высоких требований к функциональности и гибкости систем управления. Отечественным примером систем такого уровня являются «1С Бухгалтерия», «1С Предприятие», «Галактика», NS2000.

Системы высшего уровня иерархии, строящиеся на основании MRP/ERP-модели, содержат описание тысяч бизнес-процессов. Такие системы могут иметь до 100 тыс. настраиваемых параметров, позволяющих реализовать огромное многообразие требований различных предприятий. ERP-системы удовлетворяют большинству запросов как средних, так и очень крупных предприятий. Они могут работать на различных платформах: Windows NT, UNIX, Solaris, AIX и др. — с различными мощными профессиональными СУБД. Затраты на создание ERP-системы оцениваются в несколько тысяч человеко-лет. Для таких сложных информационных систем становится важным процесс апробации на множестве предприятий. Только после нескольких десятков успешных внедрений, ERP-система может претендовать на рыночный успех.

Массовый рынок демонстрирует повышенный интерес к роботизации бизнес-процессов (Robotic Process Automation, RPA). Крупные предприятия убедились в том, что частных и быстрых решений, реализованных на базе домашних разработок с использованием продуктов с открытым кодом или коммерческих платформ, недостаточно для достижения поставленных целей. Проблемы возникали с увеличением нагрузки на решение, с новыми задачами либо с увольнением разработчиков. Возникла потребность в роботизированных технологиях на базе промышленных платформ [45].

Использование роботизации бизнес-процессов началось в страховых компаниях и банках. Однако постепенно RPA приходит на предприятия реального сектора экономики (энергетика, машиностроение, нефтегаз, транспорт и логистика), вставшие на путь цифровой трансформации.

[1] Системы управления бизнесом.

Например, для нефтегазовой отрасли характерна глобализация бизнеса. Многие нефтегазовые компании построены по принципу совместных предприятий, дочерних или зависимых обществ. Как результат трансформации производства и бизнеса, возникают большие системы, требующие обработки разнообразной информации от разнородных информационных систем, созданных в разное время и на разных технологических платформах. Соответственно нужно «наводить мосты» между «островами автоматизации», что открывает новые перспективы для применения RPA, разрабатывать и вводить стандарты роботизации бизнес-процессов [42].

Выбор RPA-платформы во многом зависит не только от ее технических параметров, но и от корпоративной политики и локальной конъюнктуры рынка. Различные аналитические обзоры [38, 39] отдают высокие системам Automation Anywhere, BluePrism и UiPath. К недооцененным платформам относят платформы Kofax RPA и SAP Intelligent RPA. Компания Kofax предлагает программного робота «из коробки», интегрированного с фирменными технологиями машинного обучения, а SAP в рамках технологического стека Leonardo представляет решение Intelligent RPA, интегрированное с системами машинного обучения, интернетом вещей и аналитикой больших данных через SAP Cloud Platform [36]. Обе платформы являются финансово устойчивыми и предлагают интеграцию с другими продуктами корпоративного класса: ECM, OCR, ERP и пр.

Информационно-управляющая система XXI в. отличается от более ранних вариаций тем, что ее функционал более декомпозирован и позволяет компании поддерживать высокую степень организации данных, характеризующих ее деятельность. С экономической точки зрения ИУС — это актив компании, который создается строительством и поддерживается в эксплуатации в целях выполнения организацией хозяйственной деятельности [40, 41].

Облака являются ключевым компонентом инициатив в сфере цифровой трансформации. По прогнозам IDC, в компаниях будет доминировать мультиоблачный подход и к 2023 г. 70 % ИТ-организаций реализуют стратегические сценарии использования контейнеров и API, чтобы улучшить переносимость своих приложений в облачные среды [38, 39]. Продолжают создаваться альянсы между основными облачными провайдерами, как, например, партнерство Oracle и Microsoft по организации высокоскоростных соединений между Oracle Cloud и Azure [38, 39].

Одной из главных задач поставщиков облачных сервисов и разработчиков платформ управления облаками является укрепление безопасности приложений и данных в гибридных облачных средах, что требует присутствия на рынке средств защиты более высокого уровня, чем идентификация и управление доступом.

Интернет вещей (IoT) также является важной частью цифровой трансформации мировой экономики. Глобальные продажи оборудования для интернета вещей в 2019 г. достигли 465 млрд долл., а количество подобных устройств, находящихся в эксплуатации, выросло до 7,6 млрд шт. Такие данные привели в исследовательской компании Transforma Insights.

Наиболее популярными технологиями коммуникаций на рынке IoT-оборудования являются Wi-Fi, Bluetooth, Zigbee и 3G/4G, на которые в 2019 г. пришлось 74 % поставок продуктов. Количество соединений в публичных сетях (преимущественно в сотовых) в 2019 г. составило 1,2 млрд долл., их доля оказалась равной 16 %.

Однако распространение IoT может замедлиться в связи с недостаточной пропускной способностью сетей. Одним из способов повышения производительности IoT может быть протокол NB-IoT, который использует потенциал современных сотовых сетей. В конце января 2018 г. российская Государственная комиссия по радиочастотам выделила частоты, которые операторы связи могут использовать для стандарта NB-IoT. Это важное событие, которое открывает дорогу для доступного развертывания интернета вещей (IoT) в России.

Контрольные вопросы для самопроверки

1. Приведите примеры моделей и стандартов управления производством и производственной деятельностью. Что определяют указанные стандарты?
2. Опишите процессы интеграции ERP II-систем. На каких платформах реализуется ERP II-система?
3. Что такое роботизация бизнес-процессов? Где она используется?
4. От чего зависит выбор RPA-платформы?
5. В чем состоит экономическая сущность современной ИУС?

Раздел Б
Разработка информационно-управляющих систем

Системы реального времени — это автоматизированные системы, работа которых определяется внешними по отношению к системе, реальными временны́ми интервалами. Практически все ИУС являются системами именно реального времени. Аспекты, связанные с реальным временем, проявляются прежде всего там, где реализуется автоматическое управление объектом — на нижних трех уровнях модели (см. рис. 1). На уровне управления вводом-выводом и часто на уровне диспетчерского управления приходится использовать специализированное системное программное обеспечение — операционные системы реального времени.

5.1. Операционные системы реального времени

Работа операционных систем реального времени (ОС РВ) определяется внешними процессами, происходящими в управляемом объекте. ОС РВ обязана реагировать на события, которые могут возникать в самом управляемом объекте или быть результатом внешнего воздействия. Время ответной реакции ОС РВ строго регламентировано и определяется динамикой управляемого объекта.

С точки зрения ОС внешнее событие представляется прерыванием. Генерирование прерывания вызывает следующий (существенно упрощенный) поток событий (рис. 14):

- генерируется прерывание;
- через некоторое время T_{il}, обработчик прерывания получает управление;
- обработчик в течение интервала времени T_{int} выполняет критическую по времени работу (чтение-запись аппаратных регистров, сохранение полученной информации в памяти);
- ставится в очередь процесс, который будет осуществлять окончательную обработку прерывания;

- по завершении работы обработчика, через определенное время T_{iret} выполняется выход из прерывания;
- если обрабатывающий прерывание процесс является самым приоритетным, через некоторое время T_{sl} диспетчер передает ему управление.
- по завершении этого процесса, обработка прерывания заканчивается.

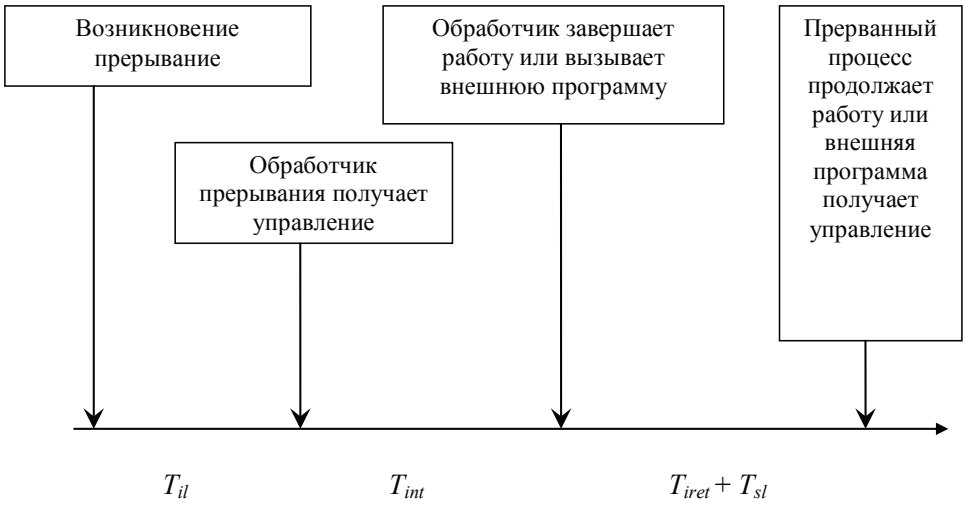

Рис. 14. Упрощенная схема потока событий, вызываемых прерыванием

В этой схеме только время обработки прерывания T_{int} зависит от содержания прерывания. Все остальные временны́е интервалы определяются прежде всего архитектурой операционной системы. Для характеристики потока событий, вызываемых прерыванием, определяют еще один временной интервал — время переключения контекста T_{cont}, который означает время передачи управления между процессами. Таким образом, мы получаем следующие характерные временные интервалы, которые определяют так называемую *реактивность* операционной системы:

- T_{il} — время задержки прерывания либо ожидания обработки прерывания (Interrupt Latency);
- T_{int} — время обработки прерывания (Interrupt Time);
- T_{iret} — время возврата из прерывания (Interrupt Termination time);
- T_{sl} — время задержки диспетчеризации (Sheduler Latency);
- T_{cont} — время переключения контекста.

Обычно соотношение временных интервалов выглядит следующим образом: $T_{sl} > T_{il} > T_{iret}$. Время переключения контекста — $T_{cont} \sim 1.2\ T_{sl}$.

При проектировании ОС РВ ставится задача максимального снижения параметров реактивности, что отличает ее от ОС общего назначения. Однако главным требованием к ОС РВ является *обеспечение ее предсказуемого поведения*. Это означает, что в любой ситуации можно рассчитать максимальную величину временного интервала задержки прерывания. Если рассчитанная величина не устраивает, переходят на аппаратуру с более высоким быстродействием. Если временные интервалы имеют очень малые значения, приходится часть операций реализовывать аппаратно, т. к. в этом случае использование чисто программных решений не представляется возможным.

Отметим, что ядро ОС РВ устроено сложнее, чем ядро ОС общего назначения, и алгоритмы его работы являются более трудоемкими. Таким образом, ОС РВ не может обеспечить меньшие по сравнению с универсальной ОС значения временных интервалов задержки прерывания. Задача ОС РВ состоит в том, чтобы обеспечить стабильность этих значений.

Для обеспечения предсказуемого поведения ОС приходится принимать целый ряд мер. Важнейшая и простейшая из них состоит во введении достаточно *большого числа приоритетов*. Если каждый процесс и каждый поток будет выполняться со своим значением приоритета, то максимальная временная задержка есть результат суммы временной задержки, вносимой обработчиком рассматриваемого прерывания, с временными задержками, вносимыми всеми более приоритетными процессами. Если же какие-либо процессы либо потоки будут выполняться с одним и тем же значением приоритета, ситуация становится непредсказуемой и заранее нельзя сказать, какой из них получит управление.

В операционной системе может наблюдаться инверсия приоритетов, т. е. явление, когда менее приоритетный процесс мешает выполнению более приоритетного процесса. Приведем пример такой ситуации.

Один из процессов обращается к системному вызову, в результате которого начинает выполняться функция ядра, для которой заблокированы прерывания. В это время возникает прерывание с более высоким приоритетом, но обработчик не может начать его обработку из-за блокировки. В результате этого более приоритетный процесс вынуж-

ден ожидать окончания менее приоритетного, т. е. возникает инверсия приоритетов.

Для универсальной ОС такая ситуация не страшна. Более того, подобное синхронное выполнение системных вызовов с блокировкой прерываний позволяет упростить код ядра и увеличить скорость его работы. Например, для ОС Windows некоторые функции графического интерфейса выполняются именно в таком синхронном режиме. Для ОС РВ же такое недопустимо — все системные вызовы должны иметь возможность прерываться.

Ситуация с блокировкой прерываний возникает также при использовании примитивов синхронизации процессов и потоков. Эти системные вызовы работают с объектами ядра, код которых, для обеспечения взаимоисключения, должен выполняться атомарно, т. е. без возможности прерывания.

Для исключения блокировки прерываний и инверсии приоритетов приходится применять ряд мер:

- необходимо обеспечить *прерываемость* любого кода ядра, в т. ч. кода объектов синхронизации;
- при выполнении кода объектов синхронизации, объекты синхронизации переопределяются, т. е. варьируются их приоритеты во избежание инверсии приоритетов [46].

Приведем пример предотвращения инверсии в ОС QNX 6.2.1 [46]. Три базовых объекта синхронизации ОС QNX — мьютекс, условная переменная и семафор — реализуются на уровне микроядра системы и создаются путем вызова системной функции:

SyncTypeCreate (unsigned type, sync_t* sync, const struct _sync_attr_t* attr);

Здесь type может принимать значения:

_NTO_SYNC_MUTEX_FREE — для создания мьютекса;

_NTO_SYNC_SEM — для создания семафора;

_NTO_SYNC_COND — для создания условной переменной.

Типы атрибутов этих объектов являются псевдонимами одного-единственного типа sync_attr_t, содержащего поле protocol, значение которого определяет, каким способом ОС будет варьировать приоритеты. По умолчанию (например, когда вместо attr передается NULL) поле protocol принимает значение PTHREAD_PRIO_INHERIT. Это означает, что ОС будет использовать протокол наследования приоритетов для предотвращения инверсии. Таким образом, все примитивы синхронизации QNX могут учитывать эффект инверсии приоритетов.

В любой развитой ОС доступ к периферийным устройствам осуществляется только через драйверы. Эти драйверы обязаны входить в состав ядра ОС, иначе они не смогут функционировать. Для ИУС характерно большое количество каналов ввода-вывода, для каждой группы которых необходим драйвер. В итоге получаем большое количество драйверов, разрабатываемых производителями устройств, которые, работая в монолитном ядре, имеют доступ ко всему адресному пространству ядра. Любой плохо написанный драйвер может, таким образом, вызвать крах всего ядра. Эта проблема успешно решается при использовании *микроядерной архитектуры ядра*, когда каждый драйвер выполняется как отдельный процесс ядра, в собственном изолированном адресном пространстве. Микроядерной называется архитектура ядра, при которой оно выполняется не в виде одного (возможно многопотокового) процесса монолитного ядра, а в виде некоторого множества процессов (каждый в своем виртуальном адресном пространстве, изолированном от доступа других процессов). Среди этих процессов ядра приходится выделять управляющий процесс — микроядро, который выполняется в привилегированном режиме и может получить доступ к адресному пространству любого процесса, реализуя функции диспетчирования, обработки прерываний, взаимодействия процессов, сетевого взаимодействия.

Кроме того, микроядерная архитектура дает исключительный уровень масштабируемости — для добавления в ядро какой-либо функциональной возможности достаточно запустить соответствующий процесс. По этой причине в ОС РВ микроядерная архитектура используется очень широко. Например, широко распространенная в мире ОС QNX [47, 50] имеет микроядерную структуру ядра.

Обратная сторона микроядерной архитектуры более сложное устройство, чем у монолитного ядра. Ядро с микроядерной архитектурой имеет бóльшие размеры, чем монолитное, и работает медленнее. Для элементарной передачи информации другой, подсистеме ядра вместо простого изменения значения переменной (в монолитном ядре) требуется использования средств межпроцессного взаимодействия (IPC). Для QNX, например, базовым средством IPC является передача синхронных сообщений, что непосредственно реализуется микроядром. В связи с этим микроядерная архитектура ядра в универсальных ОС практически не применяется. Фирма Microsoft в свое время при разработке Windows NT

пыталась использовать микроядерную архитектуру, но в итоге от нее отказалась из-за медленной работы. Микроядерое ядро GNU/HURD так и не было доведено до стабильной версии.

Управление памятью в ОС РВ отличается тем, что исключается возможность применять выгрузку данных виртуальной памяти в область свопинга (при страничном управления виртуальной памятью) и исключается применение свопинга в чистом виде (когда процесс выгружается целиком). Эти механизмы никак не согласуются с требованиями реального времени.

Еще одна особенность ОС РВ связана с их применением в промышленных системах. Для таких ИУС время их жизненного цикла будет определяться длительностью эксплуатации управляемого объекта, составляющей десятки лет и более. И несмотря на все принятые меры повышения длительности наработки на отказ, контроллеры ИУС будут постепенно выходить из строя и требовать замены. Однако новые контроллеры того же типа могут быть недоступны, т. к. они уже сняты с производства. По этой причине, в частности, используется интерпретация TIC-кода в случае применения языков программирования стандарта IEC 61131 для целевой системы (несмотря на очевидную потерю производительности).

Аналогичная ситуация имеет место с серверами и рабочими станциями диспетчерского уровня. Они постепенно выходят из строя, и мы вынуждены заменять их устройствами другого типа, с другой системой команд и другой операционной системой.

При отсутствии совместимости ОС по программному интерфейсу возникает необходимость полного перепроектирования системы, что крайне затратно, занимает большое время, требует полного цикла испытаний системы и последующей сертификации. Если новая и старая ОС являются совместимыми, то достаточно сначала перетранслировать нашу систему в новую ОС и затем ее собрать. После минимального цикла испытаний она может быть введена в эксплуатацию.

Таким образом, для промышленных систем совместимость API и соответствие их стандартам открытых систем является жизненным важным и поэтому большинство ОС РВ являются POSIX-совместимыми.

В ИУС используется большое количество разнообразных устройств с ОС РВ, как это было показано на примере многоуровневой модели системы. Они выполняют различные функции и характеризуются наличием различных ресурсов. Полностью же POSIX-совместимые опе-

рационные системы имеют заслуженную репутацию ресурсоемких. На многих контроллерах таких ресурсов нет. В этой связи приходится урезать возможности ОС. Выделяют профили прикладных контекстов реального времени (ISO/IEC ISP 15287-2), под которыми понимаются согласованные подмножества стандартов POSIX.

Минимальная система — встроенная система без виртуальной памяти, файловой системы и терминала. Поддерживает один многопотоковый процесс.

Контроллер реального времени — минимальная система, дополненная файловой системой и терминалом.

Специализированная система — встроенная система с несколькими, возможно многопотоковыми, процессами и без файловой системы.

Сформулируем основные требования к операционным системам реального времени.

1. Требуется обеспечить предсказуемый характер поведения системы и исключить инверсию приоритетов. Для этого ОС РВ должна обладать:
   - развитой системой приоритетов;
   - наследованием приоритетов в примитивах синхронизации процессов;
   - исключением свопинга.
2. Часто требуется размещение всей ОС в ПЗУ.
3. Широко используется микроядерная архитектура, позволяющая обеспечить масштабируемость и изоляцию драйверов.

Выбор ОС для решения задач автоматизации определяется многими параметрами и требованиями к ОС. Помимо указанных выше учитываются требуемые характеристики системы или изделия:

- архитектурные принципы (тип ядра и планировщика, модель многозадачности и т. д.);
- временные характеристики (время реакции на прерывание, время переключения контекста, время перепланирования и т. п.);
- поддержка реального времени;
- показатели надежности (например, среднее время восстановления после отказа);
- метрики производительности (например, пропускная способность файловой системы на данном носителе);
- необходимые технологии;

- готовые наработки и опыт компании-разработчика;
- выбранное оборудование;
- целевой рынок системы либо изделия;
- экономические ограничения проекта.

5.2. Характеристики и примеры использования систем реального времени

В табл. 5 [47–50] приведены характеристики наиболее популярных операционных систем жесткого реального времени QNX Neutrino (QNX 6) и *VxWorks*, по которым можно определиться с ОС для собственной задачи автоматизации.

Таблица 5

Сравнительные характеристики встраиваемых операционных систем, поддерживающие стандарт POSIX 1003.1

Характеристики	QNX Neutrino	Wind River VxWorks
Разработчик	Канадская компания QNX Software Systems (QSS)	Американская компания (США) Wind River
Первый выпуск	2001	1987
Тип ядра	Микроядро и взаимодействующие процессы	Микроядро
Защита памяти	Да	Настраиваемая
Реальное время	Жесткое	Жесткое
Время отклика	Единицы микросекунд	Единицы микросекунд
Количество уровней приоритета	256	256
Дисциплины планирования	FIFO, карусельная, спорадическая	Карусельная
Поддержка квотирования ресурсов процессора (ARINC 653)	Да	Да
Требуемый объем ПЗУ	Единицы мегабайтов	Сотни килобайтов
Возможность динамического восстановления при отказе	Любые системные и пользовательские процессы	Только пользовательские процессы

Продолжение табл. 5

Характеристики	QNX Neutrino	Wind River VxWorks
Поддерживаемые технологии (включая доступные через партнерскую сеть)		
Поддержка многопроцессорности	SMP, AMP	SMP, AMP
Стек IPv4	Да	Да
Cntr IPv6	Да	Да
Средства графического интерфейса	Штатная графическая оболочка, Qt, Adobe Flash Lite	Штатная графическая библиотека, Qt, пакет Tilcon Graphics Suite
Поддержка мультимедиа	Да	Да
Функции управления энергопотреблением	Да	Да
Поддержка реляционных баз данных	Да	Да
Поддержка OPC либо OPC-шлюзов	Да	Да
Поддержка эмуляторов ПЛК (Soft-PLC, МЭК 61131-3)	Да	Да
Поддержка промышленных шин	Да	Да
Инструментарий разработчика		
Интерфейс прикладного программирования (API)	POSIX	POSIX
Интегрированная среда разработки	Штатная IDE на базе Eclipse	Штатная IDE на базе Eclipse
Поддерживаемые языки программирования или моделирования	C/C++, Java, UML, Phyton, Ruby, Ada, Fortran	C/C++, Java, UML, Ada, Fortran
Поддержка JTAG-отладчиков	Да	Да
Поддерживаемые инструментальные ОС	Windows, Linux	Windows, Linux, Solaris
Поддерживаемое оборудование		
Поддерживаемые архитектуры процессоров	X86, ARM, MIPS, PowerPC	X86, ARM, MIPS, PowerPC, GoldFire

Окончание табл. 5

Характеристики	QNX Neutrino	Wind River VxWorks
Типовое применение и сертификация		
Типовое применение	Ответственные системы, авиация или космонавтика, промышленные контроллеры, военные приложения, коммуникационные устройства, медицинские приборы	Ответственные системы, авиация или космонавтика, промышленные контроллеры, военные приложения, коммуникационные устройства, медицинские приборы
Сертификаты и сертификационные пакеты	МЭК 15408 («Общие критерии») EAL 4+, МЭК 61508 SIL 3	МЭК 15408 («Общие критерии») EAL 4/4+/6+, DO-178B уровень A, МЭК 61508 SIL 3, CENELEC EN 50128, FDA 510 (k)
Лицензирование		
Доступность исходного текста	Полностью	Полностью
Лицензионные отчисления	Да	Да

Журнал Dedicated Systems, проводивший в 2011 г. сравнительное тестирование QNX RTOS 6.1 и VxWorks AE 1.1 на x86 платформе, отметил следующие существенные отличия этих систем:

- задержка при реакции на прерывание у QNX значительно выше, чем у VxWorks;
- время переключения контекста у QNX в 1,5 раза меньше, чем у VxWork;
- по стабильности временных показателей QNX опережает VxWorks при увеличении числа потоков с 2 до 128 ед., время переключения контекста у QNX выросло в 1, 65 раз, тогда как у VxWorks — в 2,24 раза.

По мнению журнала *Dedicated Systems*, ОС РВ QNX является совершенной с технической точки зрения, но по опыту применения в ответственных приложениях она проигрывает ОС РВ VxWorks. Автори-

тет VxWorks страдает из-за проводимой компанией Wind River Systems стратегии более закрытых разработок в сравнении с QSS. Обе ОС остаются пока достаточно дорогими.

Отметим, что временны́е характеристики последних версий промышленных ОС практически элиминировали существовавшую грань между «жесткими» и «мягкими» ОС РВ. Сейчас OS-9, которая ранее рассматривалась как «мягкая» ОС, по временным параметрам практически не уступает классическим «жестким» ОС — pSOS+ и VxWorks.

Наблюдается активный процесс слияния универсальных ОС и ОС реального времени. Появляются различные инструменты поддержки режима реального времени, встраиваемые в привычные операционные системы. Этот класс продуктов сочетает в себе привычный пользовательский интерфейс, средства разработки программ и API-интерфейс реального времени. Например, операционная система Linux изначально не проектировалась как система реального времени и не может быть ОС РВ по многим параметрам. Однако уже существует несколько дистрибутивов Linux, поддерживающих расширения реального времени, к ним относятся дистрибутивы компаний IBM, RT Linux от Novell и Red Hat.

В *RT Linux* [50] реализованы таймеры высокой точности: для представления системного времени теперь используется 64-битная величина, позволяющая хранить время с наносекундной точностью. Код управления таймерами и системным временем полностью переписан и теперь не допускается возникновение задержек, превышающих несколько микросекунд. Другим важным расширением стало вытесняемое ядро: к существующим в основном ядре Linux-моделям вытеснения было добавлено полное вытеснение, при котором большинство абсолютных блокировок внутри ядра заменено на семафоры [50]. Таким образом, RT Linux — это операционная система, в которой маленькое ядро реального времени сосуществует со стандартным POSIX-ядром Linux. Функции ядра и обработчики прерываний, «обернутые» в обычные системные потоки, в данной версии RT могут вытесняться другими более приоритетными потоками [50].

Функция наследования приоритетов в RT Linux позволяет решить проблему инверсии приоритетов.

Большое внимание в RT Linux уделено планированию в симметричных многопроцессорных системах. Если задача предсказуемого плани-

рования на одиночном процессоре решается легко: достаточно только выбрать самый приоритетный процесс из очереди планировщика, то в многопроцессорных системах такая очередь существует уже для каждого процессора [50]. Ядро операционной системы должно регулярно «пересматривать» очереди, чтобы обеспечить первоочередное выполнение потоков с высоким приоритетом на любом из процессоров [50].

Изменения в операционной системе требуют также аппаратной поддержки. К примеру, многие современные серверы используют прерывания по управлению системой (System Management Interrupt — SMI), связанные с ошибками оборудования и управлением питания. Эти прерывания обладают непредсказуемой латентностью и потому сложно сочетаются с работой систем реального времени. Некоторые аппаратные платформы, такие как Blade-серверы IBM, были специально модифицированы для работы с системами реального времени, и основная часть SMI-прерываний обрабатывается в BIOS и других аппаратных контроллерах в обход основной ОС.

Рассмотрим прикладной модуль, который использует ядро реального времени RT Linux в виде загружаемого модуля Linux:

```
#define MODULE
#include <Linux/module.h>
/*необходимо для задач реального времени*/
#include <Linux/rt_sched.h>
#include <Linux/rt_fifo.h>
RT_Task mytask;
```

В заголовке модуля определяется структура RT_TASK, которая содержит указатели на структуры данных модуля, а также на управляющую информацию. В нашем случае, для связи между процессами, используются очереди сообщений FIFO. Модуль реального времени читает данные из устройства и кладет их в очередь сообщений, откуда их потом забирает обыкновенное приложение Linux.

Как и каждый модуль Linux-ядра, процесс реального времени должен выполнить процедуры инициализации и завершения, аналогичные процедурам в модулях Linux:

```
int init_module (void)
{
/*определить номер очереди сообщений*/
         #define RTFIFODESC 1
```

```
/*взять локальное время*/
        RTIME now = rt_get_time ()
        rt_task_init (&mytask, mainloop,
        RTFIFODESC, 3000, 4);
        rt_task_make_periodic (&mytask,
           now+1000, 25000);
        return 0;
}
```

Подобный модульный подход к написанию приложений характерен для многих расширений реального времени. Задача реального времени работает независимо от ОС и, будучи критичной к времени реакции, программируется с помощью API-интерфейсов, предусмотренных расширением реального времени. Все служебные и интерфейсные функции возлагаются на традиционную операционную систему.

С технологической точки зрения в современных системах реального времени все чаще требуется поддержка различных коммуникационных протоколов, кластерных конфигураций, языков высокого уровня [47–50]. Более того, приложения реального времени становятся все сложнее, и использовать низкоуровневые языки программирования для их создания становится все труднее и дороже. Именно это послужило стимулом для новой разработки WebSphere Real-Time Java (WRT) — виртуальной Java-машины, основанной на технологии IBM J9, содержащей уникальную функциональность для полноценной поддержки систем жесткого и мягкого реального времени.

WRT поддерживает Java 2 Standard Edition версии 5, включая стандартную библиотеку классов Java. Виртуальная машина реализует спецификацию Java реального времени (JSR 1), обеспечивая интерфейс для потоков реального времени и различных схем управления памятью в условиях производительности реального времени. WRT поддерживает компиляцию в ходе исполнения программ в условиях реального времени и содержит набор утилит для диагностики и оптимизации кода.

Еще одно важнейшее улучшение виртуальной Java-машины — реализация спецификации Real-Time Specifications for Java (RTSJ, или JSR 1), которая расширяет функциональность Java для решения задач реального времени. В реализации RTSJ от IBM поддерживается приоритет потоков и наследование приоритетов. В WRT реализованы потоки реального времени: с поддержкой приоритетов, заданными ин-

тервалами реакции системы, доступом к внешним областям памяти и обнаружением блокировок. Существует особая возможность запуска потоков реального времени, и для них используется специальная область памяти. Для полноценной поддержки систем реального времени, в виртуальной машине Java были реализованы функции по управлению синхронизацией потоков, доступу к системному высокоточному таймеру и работе с асинхронными событиями. Многие участки кода виртуальной машины были полностью обновлены, чтобы соответствовать требуемой производительности.

Таким образом, приложения на базе WebSpere Real-Time Java могут существенно упростить создание и поддержку систем реального времени. Полное решение для систем реального времени от IBM включает в себя серверы-лезвия IBM на платформе x86-64, операционную систему реального времени RT Linux (дистрибутив IBM, Red Hat MRG или Novell SLERT), виртуальную машину и среду разработки WebSphere Real-Time Java.

Еще одно направление развития ОС РВ — управление в реальном времени группировками устройств с использованием IoT, IIoT или IoBT[1]. В классических операционных системах реального времени, как правило, таких возможностей нет.

Операционные системы для интернета вещей должны обеспечивать самостоятельность устройства, работающего в общем контексте. Каждая «вещь» получает информацию, задания и обновляет данные из общего контекста, не заботясь о наличии других устройств в том же контексте. Например, вновь подключаемое устройство, благодаря наличию доступа к контексту, сохраняющему текущий статус задачи, продолжит выполнение задачи именно с того состояния, на котором оно прекратило свою работу из-за некоторой неисправности. Это позволит обеспечить надежность, масштабирование и решение таких задач, как автономная адаптация сетей с высокой плотностью гетерогенных устройств (порядка миллиона вещей на квадратный километр).

Проект ОС РВ МАКС (операционная система реального времени для мультиагентных когерентных систем) стартовал в 2015 г. В результате реализации проекта была создана ОС реального времени, обладающая дополнительными возможностями по обеспечению взаимодействия между устройствами в распределенной среде. Концепция распределенной

[1] Internet of Battle Things.

общей памяти позволила упростить программирование параллельно функционирующих устройств: разработчик обеспечивает выполнение действий с памятью, а ОС заботится о согласованности данных на всех узлах системы, организовывает обмен данными и самостоятельно решает вопросы возможных сбоев устройств или каналов связи.

Система поставляется с комплектом русскоязычной документации, поддерживает работу с оборудованием отечественного производства (продукция «ПКК Миландр» — 32-разрядные микроконтроллеры серий 1986 и 1967). Ее техническая поддержка осуществляется российскими специалистами. К исходным кодам системы прилагаются шаблоны проектов для различных сред разработки и готовые программы, помогающие пользователям быстро освоить ОС, настраивать и создавать новые приложения.

Для обеспечения работы многозадачного приложения, ядро системы должно объединять в себе: планировщика (диспетчера), объекты синхронизации (семафоры, события и др.) и средства обеспечения взаимодействия задач (очереди сообщений).

5.3. Программный интерфейс систем реального времени

Наблюдается тенденция использования универсальных ОС в качестве ОС РВ. С одной стороны, это обусловлено тем, что ОС общего назначения широко распространены, доступны, обладают развитыми и эффективными средствами разработки. Также для них подготовлено большое количество специалистов. С другой стороны, в состав ядер универсальных ОС включаются средства поддержки реального времени, что отражается в открытых стандартах. Однако на базе такой ОС может быть реализована только система мягкого реального времени, т. к. в ней отсутствуют необходимые для ОС РВ механизмы, описанные выше.

Рассмотрим особенности использования универсальных ОС семейства Windows в качестве ОС РВ.

Система поддерживает всего 32 приоритета (0 ...31). При создании процесса указывается только класс приоритета:

- IDLE_PRIORITY_CLASS, базовый приоритет 4;

- NORMAL_PRIORITY_CLASS, базовый приоритет 9 (в фокусе) или 7;
- HIGH_PRIORITY_CLASS, базовый приоритет 13;
- REALTIME_PRIORITY_CLASS, базовый приоритет 24.

Диспетчер может самостоятельно изменять приоритет процессов всех классов, кроме класса реального времени. Класс приоритета текущего процесса задается следующим образом:

```
BOOL SetPriorityClass (
HANDLE hProcess,//дескриптор процесса
DWORD dwPriorityClass//значение класса приоритета
);
```

При создании потока он принимает приоритет, равный базовому приоритету процесса, и далее он может быть изменен в диапазоне (±2) относительно базового или принимает крайние значения приоритета класса (16 или 31 для класса реального времени):

- THREAD_PRIORITY_LOWEST
- THREAD_PRIORITY_BELOW_NORMAL
- THREAD_PRIORITY_NORMAL
- THREAD_PRIORITY_ABOVE_NORMAL
- THREAD_PRIORITY_HIGHEST
- THREAD_PRIORITY_IDLE
- THREAD_PRIORITY_TIME_CRITICAL

Приоритет потока задается следующим образом:

```
BOOL SetThreadPriority (
HANDLE hThread,//дескриптор потока
i nt nPriority//уровень приоритета потока
);
```

Таким образом, общее количество приоритетов равно 7, что является крайне недостаточным.

При обработке прерываний в Windows происходят следующие события:

- прерывание;
- сохранение регистров и вызов диспетчера;
- сохранение контекста;
- выполнение ISR (Interrupt Service Routine) — критическая по времени работа;
- постановка в очередь DPC (Deferred Procedure Call);
- восстановление контекста и регистров;

- выполнение DPC с приоритетом DISPATCH_LEVEL — окончательная обработка прерывания;
- продолжение выполнения пользовательских процессов.

Обработка прерывания подобна процедуре, рассмотренной выше, за исключением того, что все процедуры DPC, представляющие собой части драйвера (как и ISR), выполняются с одним и тем же самым высоким в системе приоритетом.

Таким образом, для Windows характерна постоянно проявляющаяся инверсия приоритетов. В системе часто возникают непредсказуемые временные задержки. Особенно это связано с передачей данных по сети, т. к. DPC сетевого драйвера отрабатывает немалый тайм-аут, что составляет секунды. В это время все остальные процессы заблокированы.

Системы Linux, являясь представителями семейства UNIX и, следовательно, POSIX-совместимыми, гораздо полнее отвечают требованиям реального времени [48, 49]. Обработка прерываний в Linux полностью соответствует рассмотренной выше процедуре.

В Linux присутствуют приоритеты нескольких типов. Базовый приоритет процесса имеет 40 значений. Он принимает значения от –20 — самое высокое, до +19 — самое низкое. Поскольку самым приоритетным является процесс с минимальным значением приоритета, в литературе базовый приоритет иногда называется «любезностью». Подразумевается, что процесс, имеющий высокое значение приоритета, любезно разрешает себя прерывать.

Установка базового приоритета процесса осуществляется следующим образом:

```
int setpriority (int which, int who, int prio);
which:
• PRIO_PROCESS
• PRIO_PGRP
• PRIO_USER
who:
• PID
• PGID
• UID
prio: -20..19
```

В программном коде указывают, для какого объекта задается приоритет, идентификатор объекта и сам приоритет. Базовый приори-

тет используется диспетчером для принятия решения о предоставлении процессу кванта времени и для вычисления начального значения этого кванта.

Планирование процессов осуществляется диспетчером в соответствии с заданной для процесса или потока дисциплиной диспетчеризации. Выделяют следующие дисциплины диспетчеризации:

- очередь — FIFO (First In First Out). Процесс из очереди готовых к выполнению процессов данного приоритета получает управление и выполняется до тех пор, пока не завершится;
- карусель — Round Robin. Процессу выделяется квант времени, который фиксирован или является заранее вычисляемой величиной. По истечении кванта времени процесс теряет управление и помещается в конец очереди готовых к выполнению процессов данного приоритета;
- адаптивный алгоритм;
- спорадический алгоритм.

Адаптивный алгоритм диспетчеризации работает следующим образом:

- из очереди готовых к выполнению процессов выбирается процесс с максимальным приоритетом, который находится в очереди самое длительное время. Ему выделяется квант времени фиксированного размера;
- по истечении кванта времени приоритет процесса начинает линейно уменьшаться;
- когда динамический приоритет процесса становится меньше приоритета какого-либо процесса в очереди, выполняется переключение. Процесс помещается в очередь с исходным значением приоритета.

Эта дисциплина использовалась в классических системах UNIX и в QNX4. В Linux не применяется.

Спорадический алгоритм диспетчеризации работает следующим образом. Процесс имеет 2 приоритета: основной N (normal) и фоновый L (foreground). Когда диспетчер выбирает процесс для выполнения, происходит следующее:

- при запуске поток имеет начальный бюджет времени C (initial budget) и основной приоритет N;
- если время обработки, заданное бюджетом времени C, заканчивается, приоритет потока понижается до фонового значения L;

- через период времени T происходит пополнение (replenishment) бюджета времени потока до значения C, и он снова может выполняться с основным приоритетом N.

Этот алгоритм широко используется, например, в QNX6 [46, 47] и в последних ядрах Linux [48, 49].

При планировании выделяют 2 типа процессов:

- обычные (интерактивные и пакетные) процессы;
- процессы реального времени.

Для диспетчеризации обычных процессов используется статический приоритет s, описываемый функцией nice (). Для всех готовых к выполнению процессов вычисляется динамический приоритет d: в зависимости от времени ожидания процессов, диспетчер может изменить приоритет процесса до значения $d = s - b + m$ (m — целое число, например 5 или 20, $b = [10 \cdot t]$, где t — среднее время, проведенное процессом в ожидании за 1 с и выраженное в секундах), но ограниченного максимальным значением, задаваемым системой. Из списка готовых к выполнению процессов выбирается процесс с минимальным значением динамического приоритета. В зависимости от статического приоритета, вычисляется величина базового кванта t_q как функция s. Процесс выполняется в течение интервала времени, соответствующего базовому кванту t_q.

Для диспетчеризации процессов реального времени используется статический приоритет реального времени. Стандарт POSIX требует не менее 32 значений этого приоритета. ОС Linux поддерживает 99 значений. Диапазон значений статического приоритета реального времени может быть получен при помощи функций:

```
int sched_get_priority_min (int policy);
int sched_get_priority_max (int policy);
```

где `policy` — дисциплина диспетчеризации.

Дисциплина диспетчеризации для процесса и его статический приоритет реального времени могут быть заданы следующим образом:

```
int sched_setscheduler (pid_t pid, int policy, const
struct sched_param *p);
struct sched_param {… int sched_priority; …};
```

где pid — идентификатор процесса, pid=0 означает текущий процесс;

policy — дисциплина диспечеризации;

sched_priority — статический приоритет реального времени (1 ...99).

Поддерживаются следующие символические константы для обозначения дисциплин диспечеризации:

- SCHED_FIFO — First in-first out scheduling;
- SCHED_RR — Round-robin scheduling;
- SCHED_DEADLINE — Sporadic task model deadline scheduling (since 3.14);
- SCHED_OTHER (sched_priority=0);
- SHED_BATCH (sched_priority=0);
- SHED_IDLE (sched_priority=0).

В конкретных системах часть из них может отсутствовать. Обязательно поддерживаются SCHED_FIFO, SCHED_RR, SCHED_OTHER. Дисциплины диспетчеризации SCHED_OTHER, SHED_BATCH и SHED_IDLE применяются только для обычных низкоприоритетных процессов, статический приоритет реального времени которых всегда равен 0.

Процесс может самостоятельно обратиться к диспетчеру и досрочно прервать квант времени

```
int sched_yield (void);
```

Обычно это используется применительно к дисциплине SCHED_FIFO для реализации невытесняющей многозадачности.

В случае использования дисциплины SCHED_RR, диспетчер вычисляет величину кванта времени по значению динамического приоритета процесса. Получить значение этого кванта можно следующим образом:

```
int sched_rr_get_interval (pid_t pid, struct timespec
*tp);
struct timespec {
time_t tv_sec;/* секунды */
long tv_nsec;/* наносекунды */
};
```

Приведем пример программы, определяющей величину кванта времени:

Листинг kvant.c

```
#include <stdio.h>
#include <unistd.h>
#include <sys/time.h>
#include <sys/resource.h>
```

```
#include <sched.h>
#include <errno.h>
__uint64_t timespec2nsec (struct timespec *tp)
{
return (__uint64_t) (tp->tv_sec)*1000000000+/* seconds */
tp->tv_nsec;/* nanoseconds */
}
int main ()
{
struct timespec ts;
__uint64_t kv;
int i;
struct sched_param sp;
sp.sched_priority=0;
sched_setscheduler (0, SCHED_RR,&sp);
for (i=-20; i<=20; i++) {
errno=0;
printf ("%d\t%d\t%d\t", i, setpriority (PRIO_PROCESS,0,
i), errno);
sleep (1);
sched_rr_get_interval (0, &ts);
kv=timespec2nsec (&ts);
printf ("%lu\n", (unsigned long)kv/1000);
}
}
```

Программа позволяет исследовать зависимость кванта времени от базового приоритета, приоритета реального времени и дисциплины диспетчеризации.

Современные операционные системы в обязательном порядке должны позволять создавать многопотоковые процессы. При создании потока задается функция, которая будет выполняться в потоке, структура атрибутов и адрес, куда будет помещен результат работы потока:

```
int pthread_create (
pthread_t *thread,//переменная для записи идентификатора потока TID
const pthread_attr_t *attr,//структура атрибутов потока
```

```
  void * (*start_routine) (void *),// – функция, выполняемая в потоке
  void *arg);// – результат работы потока
```

Атрибуты потока задаются в структуре, которая сначала инициализируется

```
  int pthread_attr_init (pthread_attr_t *attr);
```

а затем для потока может быть задана дисциплина диспетчеризации

```
  int pthread_attr_setschedpolicy (thread_attr_t *attr, int policy);
```

и приоритет

```
  int pthread_attr_setschedparam (pthread_attr_t *attr, const struct sched_param *param);
  struct sched_param {… int sched_priority; …};
```

По умолчанию поток создается как присоединенный, при этом возможна его синхронизации с родительским потоком.

Ожидание завершения присоединенного потока имеет вид:

```
  int pthread_join (pthread_t thread, void **value_ptr);
  value_ptr – указатель результата работы потока.
```

Отсоединение потока — операция безвозвратная и синхронизация невозможна:

```
  int pthread_detach (pthread_t thread);
```

Завершается поток либо путем простого завершения выполнения функции в нем, при этом можно указать числовой результат работы потока

```
  return (void *)data;
```

либо путем вызова функции

```
  int pthread_exit (void *value_ptr);
```

где value_ptr — указатель области данных, передаваемых родителю (результат выполнения потока).

Таким образом может быть построена иерархия потоков: главный поток (с функцией `main ()`), его дочерние потоки, дочерние потоки дочерних потоков и т. д. Процесс завершается по окончании главного потока, и для успешного выполнения всех потоков необходима их синхронизация (ожидание завершения дочерних потоков).

Приведем пример демонстрационной программы, которая создает несколько потоков, задает для них дисциплину диспетчеризации и приоритеты, определяет величину кванта времени.

Листинг Pth_kvant.c

```
  #include <pthread.h>
  #include <string.h>
  #include <stdio.h>
  #include <stdlib.h>
  #include <unistd.h>
  #include <errno.h>
  #include <ctype.h>
  struct thread_info {/* Used as argument to thread_start
() */
  pthread_t thread_id;/* ID returned by pthread_create ()
*/
  int thread_num;/* Application-defined thread # */
  char *argv_string;/* From command-line argument */
  };
  __uint64_t timespec2nsec (struct timespec *tp)
  {
  return (__uint64_t) (tp->tv_sec)*1000000000+/* seconds */
  tp->tv_nsec;/* nanoseconds */
  }
  struct timespec ts;
  /* Thread start function */
  static void *thread_start (void *arg)
  {
  struct thread_info *tinfo = arg;
  char *uargv, *p;
  __uint64_t kv;
  sched_rr_get_interval (0, &ts);
  kv=timespec2nsec (&ts);
  printf ("Thread%d: argv_string=%s; Time kvant:%lu\n",
  tinfo->thread_num, tinfo->argv_string, (unsigned long)
kv/1000);
  uargv = strdup (tinfo->argv_string);
  if (uargv == NULL) perror ("strdup");
  return uargv;
  }
  int main (int argc, char *argv [])
  {
```

```
  int s, tnum, opt, num_threads;
  struct thread_info *tinfo;
  pthread_attr_t attr;
  int stack_size;
  void *res;
  struct sched_param param;
  int policy=SCHED_RR;

  num_threads = argc;
  /* Initialize thread creation attributes */
  s = pthread_attr_init (&attr);
  if (s!= 0) perror ("pthread_attr_init");
  /* Allocate memory for pthread_create () arguments */
  tinfo = calloc (num_threads, sizeof (struct thread_
info));
  if (tinfo == NULL) perror ("calloc");
  /* Create one thread for each command-line argument */
  for (tnum = 1; tnum < num_threads; tnum++) {
  tinfo [tnum].thread_num = tnum;
  tinfo [tnum].argv_string = argv [tnum];
  s=pthread_attr_setschedpolicy (&attr, policy);
  if (s!= 0) perror ("pthread_attr_setschedpolicy");
  if (tnum==1) param.sched_priority=sched_get_priority_
max (policy);
  else param.sched_priority= (tnum+1)*10;
  s=pthread_attr_setschedparam (&attr, &param);
  if (s!= 0) perror ("pthread_attr_setschedparam");
  //The pthread_create () call stores the thread ID into
corresponding element of tinfo []
  s = pthread_create (&tinfo [tnum].thread_id, &attr,
  &thread_start, &tinfo [tnum]);
  if (s!= 0) perror ("pthread_create");
  }
  //Destroy the thread attributes object, since it is no
longer needed
  s = pthread_attr_destroy (&attr);
  if (s!= 0) perror ("pthread_attr_destroy");
  //Now join with each thread, and display its returned value
```

```
for (tnum = 1; tnum < num_threads; tnum++) {
s = pthread_join (tinfo [tnum].thread_id, &res);
if (s!= 0) perror ("pthread_join");
printf ("Joined with thread%d; returned value was%s\n",
tinfo [tnum].thread_num, (char *) res);
free (res);/* Free memory allocated by thread */
}
free (tinfo);
exit (EXIT_SUCCESS);
}
```

В демонстрационной программе создаются потоки, количество которых равно числу аргументов командной строки, им передаются эти аргументы, и они же возвращаются как результирующие значения. Для каждого потока задается дисциплина диспетчеризации и приоритет реального времени и определяется величина кванта времени.

Взаимодействие процессов и потоков в ОС РВ может осуществляться стандартным образом [47, 48]. В то же время существуют механизмы, предназначенные для применения именно в ОС РВ. С 1996 г. в рамках POSIX одно из таких средств было стандартизировано, как *сигналы реального времени*.

Сигналы реального времени, как и обычные сигналы, представляют собой простейшие быстро работающие асинхронные сообщения, передаваемые между процессами. Выделим их отличительные особенности:

- реализуется очередь сигналов, в которых экземпляры одноименных сигналов не объединяются;
- порядок доставки сигналов определяется по их номерам; сигналы с большими номерами более приоритетны (однако стандарт POSIX утверждает обратное);
- номера сигналов от SIGRTMIN до SIGRTMAX (стандартно не менее 8, в Linux их 31);
- с сигналом передается 32 бит данных;
- для работы с сигналами добавлено несколько дополнительных функций.

Действия процесса по обработке сигнала задаются, как обычно — системным вызовом sigaction:

```
int sigaction (int signum, const struct sigaction *act,
struct sigaction *oldact);
```

В параметрах указывается номер сигнала, задаваемые в структуре типа sigaction действия по обработке сигнала, адрес аналогичной структуры, в которой сохраняются предыдущие действия по его обработке. Структура типа `sigaction` теперь несколько изменена:

```
struct sigaction {
void (*sa_handler) (int);
void (*sa_sigaction) (int, siginfo_t *, void *);
sigset_t sa_mask;
int sa_flags;
void (*sa_restorer) (void);
};
```

Она имеет дополнительное поле `sa_sigaction`, в котором задается обработчик сигнала реального времени. Если мы его используем, необходимо задать опцию (sa_flags) SA_SIGINFO. Прототип обработчика выглядит следующим образом:

```
void handler (int sig, siginfo_t *info, void *ucontext)
{…}
```

где sig — номер сигнала;

info — указатель на структуру siginfo_t;

ucontext — указатель на структуру типа **ucontext_t**, содержащую информацию из сохраненного контекста процесса.

Первый параметр содержит номер сигнала, получение которого послужило причиной вызова обработчика. Второй параметр указывает на структуру, в которую записывается детальная информация о сигнале:

```
typedef struct {
int si_signo;/* Signal number */
int si_code;/* Signal code */
sigval_t si_value;/* Signal value */
…
} siginfo_t;
```

Здесь поле si_code содержит условный код, определяющий механизм генерации сигнала. Например, это может быть:

SI_USER — сигнал сгенерирован функцией kill ();

SI_KERNEL — сигнал сгенерирован ядром ОС;

SI_QUEUE — сигнал сгенерирован функцией sigqueue ().

Поле **si_value** содержит значение, передаваемое с сигналом. Оно имеет следующий тип:

```
union sigval {
int sival_int;
void *sival_ptr;
};
```

Обычно используется sival_int — целое 32-битное значение.

Отправка сигнала в очередь осуществляется по следующей функции:

```
int sigqueue (pid_t pid, int sig, const union sigval value);
```

где pid определяет процесс, которому передается сигнал;

sig — номер сигнала;

value — значение сигнала.

Первые два параметра соответствуют параметрам системного вызова kill (). Дополнительный, третий, параметр позволяет задать значение, передаваемое с сигналом.

Приведем пример программы, иллюстрирующей передачу и прием сигналов реального времени:

Листинг Sig-RT.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>

static void handler (int signo, siginfo_t* info, void*
context)
{
printf ("Received signal %d, code %d, val %d\n",
signo, info->si_code, info->si_value.sival_int);
}
int main (int argc, char *argv [])
{
printf ("SIGRTMAX= %d, SIGRTMIN= %d\n", SIGRTMAX,
SIGRTMIN);
int opt, val, beg=SIGRTMAX, num=3, fin=SIGRTMAX-num,
seg=3;
int i, j;
while ((opt=getopt (argc, argv,"b: e: n:"))!=-1) {
```

```
  switch (opt) {
  case 'b':
  if (sscanf (optarg,"%d",&val)!=1) {
  perror ("Parse"); exit (EXIT_FAILURE);
  }
  beg=val;
  break;
  case 'e':
  if (sscanf (optarg,"%d",&val)!=1) {
  perror ("Parse"); exit (EXIT_FAILURE);
  }
  fin=val;
  break;
  case 'n':
  if (sscanf (optarg,"%d",&val)!=1) {
  perror ("Parse"); exit (EXIT_FAILURE);
  }
  seg=val;
  break;
  default: exit (EXIT_FAILURE);
  }}
  num=fin-beg;
  fin+=num>0?1:-1;
  sigset_t sigset;
  sigemptyset (&sigset);
  for (i=beg; i!=fin; i+= (num>0?1:-1)) sigaddset (&sigset,
i);
  pid_t pid;
  if ((pid=fork ())==0) {
  //Child Process
  sigprocmask (SIG_BLOCK,&sigset, NULL);
  for (i=beg; i!=fin; i+= (num>0?1:-1)){
  struct sigaction act, oact;
  sigemptyset (&act.sa_mask);
  act.sa_sigaction=handler;
  act.sa_flags=SA_SIGINFO;
  if (sigaction (i,&act, NULL)<0) perror ("set handler");
  }
```

```
printf ("Child: signal mask set\n");
sleep (3);
printf ("Child: signal unblock\n");
sigprocmask (SIG_UNBLOCK,&sigset, NULL);
sleep (3);
exit (EXIT_SUCCESS);
}
//Parent process
sigprocmask (SIG_BLOCK,&sigset, NULL);
sleep (1);
union sigval value;
for (i=beg; i!=fin; i+= (num>0?1:-1)) {
for (j=0; j<seg; j++) {
value.sival_int=j;
sigqueue (pid, i, value);
printf ("Signal sent:%d, val=%d\n", i, j);
}
}
printf ("Parent finished!\n");
exit (EXIT_SUCCESS);
}
```

В программе создается новый процесс, которому родительский процесс передает группы сигналов реального времени. С помощью опций **-b** и **-e** задаются начальный и конечный номера сигналов, а с помощью опции **-n** задается количество сигналов в группе. Передаваемое значение соответствует номеру сигнала в группе. В программном коде показан порядок доставки, а также отсутствие объединения сигналов и передача с ними информации.

Контрольные вопросы для самопроверки

1. Что такое POSIX-совместимая ОС? Приведите примеры.
2. Как определяется выбор ОС для решения задач автоматизации?
3. Сравните две POSIX-совместимые ОС РВ QNX и VxWorks. Укажите их преимущества и недостатки. В чем проигрывает ОС РВ QNX

операционной системе VxWorks? По каким характеристикам ОС РВ QNX выигрывает у операционной системы РВ VxWorks?

4. Каким образом происходит развитие слияния универсальных ОС и ОС реального времени?

5. Укажите особенности дистрибутивов Linux, поддерживающих расширения реального времени.

6. Приведите пример прикладного модуля, использующего ядро реального времени RTLinux в виде загружаемого модуля Linux.

7. Что такое виртуальная Java-машина?

8. Какие функции реализованы в виртуальной Java-машине для полноценной поддержки систем реального времени?

9. Требуются ли ОС РВ для управления в реальном времени группировками устройств с использованием IoT, IIoT или IoBT? Могут ли быть с этой целью использованы ОС РВ QNX и VxWorks?

10. Что такое ОС РВ МАКС? Какие задачи она выполняет? В чем состоит концепция указанного проекта?

11. В чем сущность процесса стандартизации ОС РВ? Какое влияние оказывает этот процесс на развитие технологий автоматизированных систем управления производством?

12. Что такое реактивность операционной системы реального времени? Как рассчитать максимальную величину временного интервала задержки прерывания?

13. Что такое инверсия приоритетов? Как инверсия приоритетов влияет на предсказуемость поведения ОС?

14. Что такое наследование приоритетов? В каких целях оно используется?

15. Какая нежелательная ситуация может взникнуть при использовании примитивов синхронизации процессов и потоков?

16. Какие задачи выполняет микроядерная архитектура ядра? Опишите проблемы использования микроядерной архитектуры.

17. Что такое профили прикладных контекстов реального времени?

18. Укажите особенности использования универсальных ОС в качестве ОС РВ.

19. Назовите алгоритмы диспетчеризации, используемые в универсальных ОС и ОС РВ.

20. Приведите пример программы, позволяющей исследовать зависимость кванта времени от базового приоритета, приоритета реального времени и дисциплины диспетчеризации.

6.1. Разработка системы мониторинга электродвигателей

В качестве примера ИУС приведем реализацию удаленного мониторинга электродвигателей, оснащенных интеллектуальными приводами. В комплект привода входил модуль, который предоставлял информацию о параметрах работы двигателя по протоколу ModBus. Первым шагом была создана платформа интернета вещей IoT Hub в облачном сервисе Microsoft Azure и цифровой двойник граничного устройства — Gateway с Azure IoT Edge. Само граничное устройство представляло собой Intel Up Board с CPU Atom x5-Z8350, eMMC на 16 Гбайт и RAM на 1 Гбайт.

Цифровой двойник — это структура данных, которая содержит в себе полное описание всех настроек устройства, хранящееся в облаке. Двойник есть как у самого устройства, так и у каждого модуля — прикладного алгоритма, упакованного в контейнер Docker.

Для развертывания Gateway была установлена ОС Ubuntu версии 18.04. Дополнительно на Gateway были установлены Docker, Python и собственно среда выполнения IoT Edge, запускаемая командой `pip install -U azure-iot-edge-runtime-ctl`. Далее устройство Gateway было зарегистрировано в облачном сервисе с указанием его SAS-ключа в команде `iotedgectl setup`. После запуска контейнера управление перешло к IoT Edge Gateway уже через облако с использованием цифрового двойника.

Для передачи данных с привода в облако разворачивали модуль поддержки протокола ModBus TCP с указанием адреса его образа — microsoft/azureiotedge-modbus-tcp:1.0-preview. Для загрузки контейнера из собственного репозитория Docker необходимо было бы указать его адрес. Все адреса подробно прописаны в документации на сайте. После этого на Gateway runtime самостоятельно начали «приезжать» контейнеры с алгоритмами поддержки протокола ModBus TCP и передачи данных в облачный IoT Hub. Статус выполнения каждого кон-

тейнера отслеживался как в интерфейсе облака, так и непосредственно на контроллере.

Настройки для подключения к приводу по ModBus TCP задавались уже в облаке. Для этого заполнялась структура данных в формате JSON. Далее среда исполнения на граничном устройстве получала уведомление об изменении настроек модуля ModBus, которые синхронизировались с устройством. В результате модуль подключался к приводу и начинал передавать в облако значения регистров ModBus в формате JSON.

Настройка цифрового двойника модуля ModBus TCP выглядит следующим образом:

```
{
«DisplayName»: «RPM», «HwId»: «ABB-ACS50-0001»,
«Address»: «40001»,
«Value»: «1150»,
«SourceTimestamp»: «2018-04-2510:40:38»
},
{
«DisplayName»: «Power»,
«HwId»: «ABB-ACS50-0001»,
«Address»: «40002»,
«Value»: «5789»,
«SourceTimestamp»: «2018-04-2510:40:38»
},
{
«DisplayName»: «Amperage»,
«HwId»: «ABB-ACS50-0001»,
«Address»: «40004»,
«Value»: «4657»,
«SourceTimestamp»: «2018-04-2510:40:38»
}
```

После настройки необходимо:

- загрузить на граничное устройство модуль Azure Function и настроить преобразование значений регистров ModBus в реальные параметры двигателя с соответствующими единицами измерения;
- загрузить модуль Stream Analytics и настроить его таким образом, чтобы он отправлял не все значения регистров ModBus, а только те из них, которые отличаются от предыдущих;

- загрузить модуль машинного обучения Azure Machine Learning и настроить модель таким образом, чтобы она определяла «аномальное» сочетание параметров двигателя. Саму модель взять из галереи на Azure Marketplace;
- подключить телеметрию к решению — с открытым исходным кодом *Azure IoT Suite* с панелями мониторинга и уведомлений.

6.2. Разработка системы на базе модели промышленного контроллера под управлением ISaGRAF

Рассмотрим пример программирования контроллера на языках стандарта IEC6-1131. На контроллере в этом случае должна выполняться целевая задача ISaGRAF, а тип контроллера может быть любым, поэтому для приобретения навыков программирования[1] достаточно использовать среду разработки и моделировать выполнение программы.

В качестве примера системы на базе промышленного контроллера рассмотрим задачу программирования простейшего контроллера светофора. Светофор представляет собой устройство из трех сигнальных источника света красного, желтого и зеленого цветов. В программе они будут представлены как выходные переменные, связанные с соответствующими каналами логического вывода. Контроллер будет работать в двух режимах:

- автоматическом — сигнальные огни зажигаются по времени;
- ручном — постоянно горит зеленый свет для автотранспорта. Если пешеход собирается переходить, он нажимает специальную кнопку, и контроллер отрабатывает программу перехода — зажигает для автомобилей последовательно с задержкой желтый, красный, желтый, зеленый огни.

Кнопки управления в программе будут представляться входными переменными, связанными с соответствующими каналами логического ввода. Как любая управляющая программа, наша программа будет циклической, постоянно проверяющей значение входных логических переменных.

[1] Языки программирования описаны в книге [47].

Для создания примера будем использовать пакет ISaGRAF 6.4, в котором среда разработки называется «Единая платформа автоматизации» (Automation Collaborative Platform).

Запустив среду разработки (выполняется под управлением ОС Windows), прежде всего создаем новый проект. Для этого выбираем пункт меню File → New → Project. Появляется окно, в котором можно выбрать один из нескольких шаблонов проекта. Поскольку реальная аппаратура (контроллер) отсутствует, выбираем шаблон ISaGRAF5 → Simulator и заполняем поле имени проекта (пусть будет Svetofor). Нажимаем кнопку OK.

Появляется окно Deployment.isadpl, выбираем SIMULATOR и в Solution Explorer разворачиваем дерево проекта.

Создаем необходимые переменные, отражающие в программе каналы ввода-вывода.

В Solution Explorer щелкаем Svetofor → Device1 → Global Variables — раскрывается окно Resource1-VAR (рис. 15). В этой таблице пока отображаются только служебные переменные.

Создаем новые переменные (добавляем их в нижнюю строчку):

Red типа BOOL — выходная с атрибутом *Write* (красный сигнал светофора);

Yellow типа BOOL — выходная с атрибутом *Write* (желтый сигнал светофора);

Green типа BOOL — выходная с атрибутом Write (зеленый сигнал светофора);

Start типа BOOL — входная с атрибутом *Read* (кнопка запуска светофора);

Auto типа BOOL — входная с атрибутом *Read* (кнопка выбора автоматического режима);

Button типа BOOL — входная с атрибутом *Read* (кнопка пешехода).

Можно и в комментарии указать, что означает декларированная переменная (рис. 15).

Теперь надо связать созданные переменные с каналами ввода-вывода.

Нажимаем на Device1 — разворачивается одноименное окно. Выбираем Device1 → Resource1, в контекстном меню выбираем I/O Device — разворачивается одноименное окно. В этом окне выбираем Add device — раскрывается окно Device Selector. Выбираем симулированный модуль булевского ввода (Simulation Boolean Input device), далее выбираем коли-

чество каналов (4 — у реальных модулей количество каналов обычно кратно степени 2). В нем Выбираем симулированный модуль булевского вывода (Simulation Boolean Output device), затем — количество каналов (также 4).

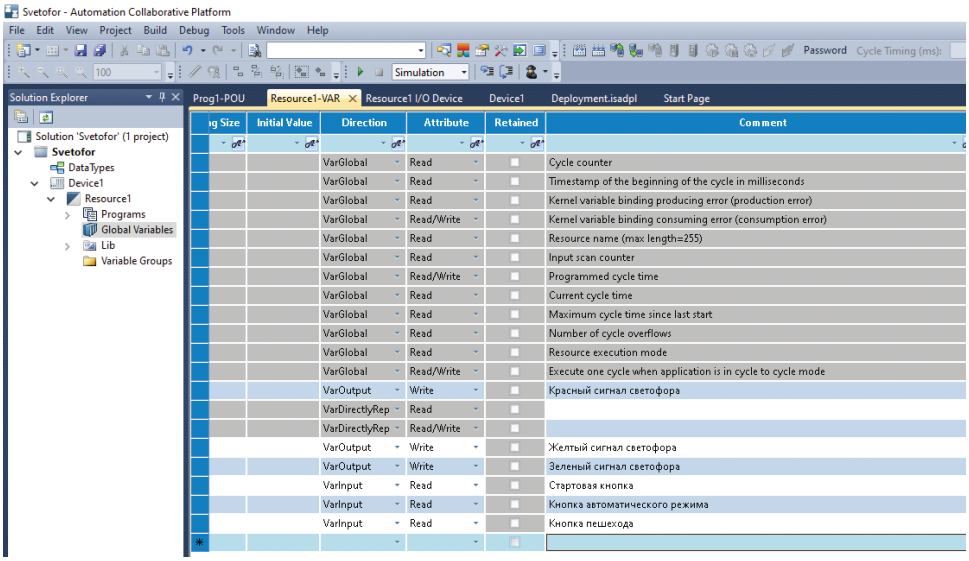

Рис. 15. Создание новых переменных

Выбранные затем устройства (после нажатия кнопки OK) появляются в дереве объектов окна I/O Device. В правой части окна отображаются также связанные с каналами этих устройств переменные прямого представления

%IX0.0, %IX0.1, %IX0.2, %IX0.3, %QX1.0, %QX1.1, %QX1.2, %QX1.3

Последовательно выбираем каждую переменную (которой соответствует канал ввода-вывода) и нажимаем безымянную кнопку в правом верхнем углу окна. Раскрывается окно Variable Selector со списком переменных. Выбираем подходящую переменную, нажимаем кнопку OK — переменная оказывается связанной с выбранным каналом и отображается в списке (рис. 16).

Эти действия повторяем для каждой декларированной переменной. Теперь в списке переменных (*Resource1-VAR*), в колонке *Wiring*, отображается связанный с переменной канал в виде имени переменной прямого представления.

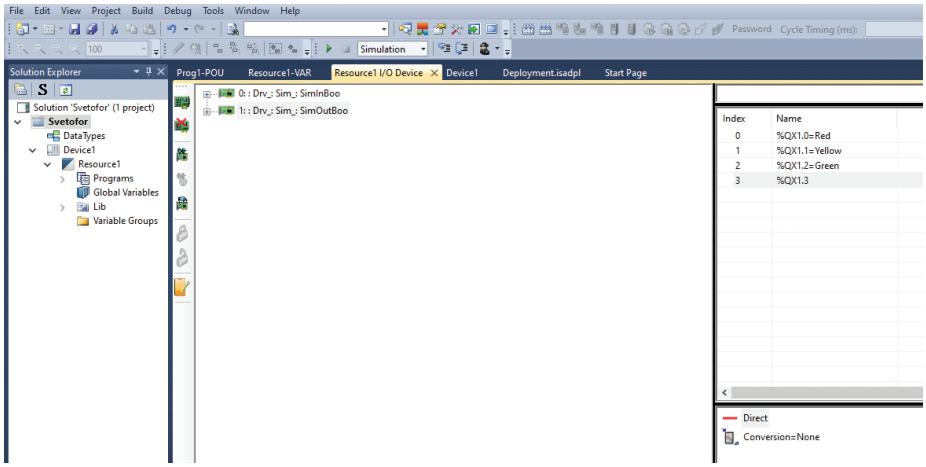

Рис. 16. Связь переменных с каналами ввода-вывода

Далее напишем программу, которая и будет выполняться на нашем контроллере. Для этого в Solution Explorer выбираем Svetofor → Device1 → Resource1 → Programms, указываем контекстное меню и выбираем в нем пункт Add → New SFC: Sequential Function Chart. Создается новый объект Prog1, выбирая который, мы попадаем в новое окно редактора SFC-программ Prog1-POU, в котором и набираем нашу программу, используя элементы конструктора, отображаемые в окне Toolbox.

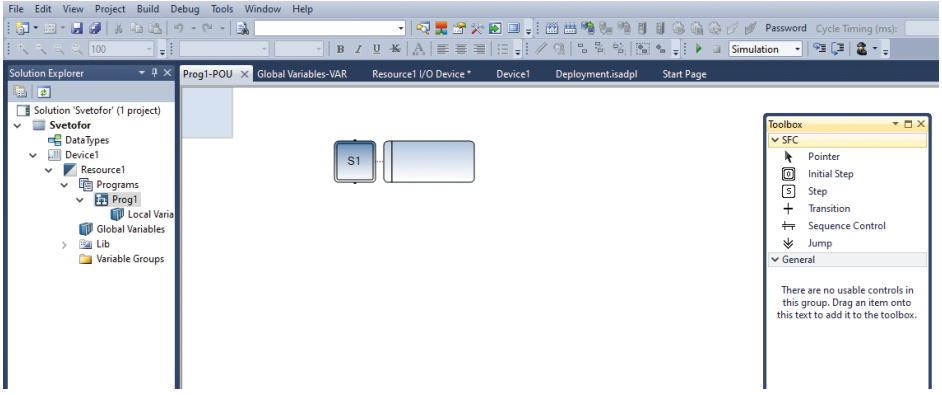

Рис. 17. Создание программы с помощью конструктора

Экранная форма начала программы показано на рис. 17. В начальном блоке устанавливаем значение «ложь» для всех выходных переменных (т. е. гасим все сигнальные лампы светофора).

Шаг 1: S1 выполняется, когда значение переменной Start — «истина» (нужно нажать стартовую кнопку).

Шаг 2: S2 является буферным и не содержит какие-либо действия.

Далее имеет место ветвление с помощью одиночного расхождения (рис. 18), т. е. выполняется либо левая ветвь шагов, либо правая ветвь шагов, в зависимости от того, нажата или нет кнопка пешехода (переменная Button): если кнопка пешехода не нажата, то выбирается левая ветвь, если кнопка нажата, то выбирается правая ветвь.

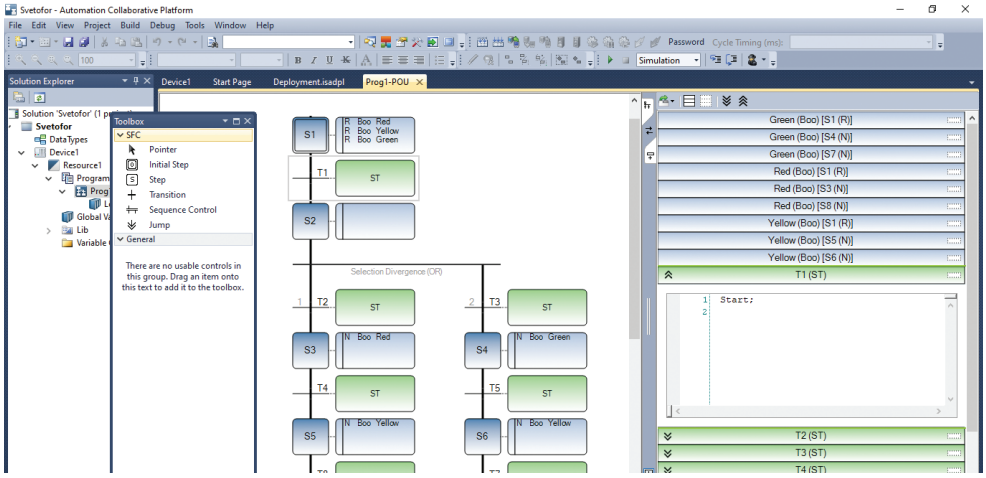

Рис. 18. Разработка программы

Переходы между шагами левой ветви ветвления S3, S5, S7 выполняются в автоматическом режиме (auto — «истина»). Это соответствует автоматическому переключению сигнальных ламп светофора — «красный» (шаг S3 — Red), «желтый» (шаг S5 — Yellow), «зеленый» (шаг S7 — Green). Процесс переключения осуществляется циклически. На шаге S3 зажигается красный свет и горит 10 с. Переход T4 выполняется, если время выполнения шага 3 превосходит 10 с. Это означает, что после 10-секундного горения S1 гасится красная сигнальная лампа и включается желтая сигнальная лампа S5. Временная переменная перехода T6, равная также 10 с, определяет время горения S5. Это означает, что через 10 с горения S5 эта сигнальная лампа будет выключена, а зеленая сигнальная лампа S7 будет включена. S7 горит 10 с.

Правая ветвь ветвления может быть включена в ручном режиме. Для этого необходимо нажатие кнопки пешеходом. Здесь зажигается

зеленый свет (шаг S4) и горит, пока пешеход еще раз не нажмет кнопку (переменная Button). Затем происходит переход светофора в режим горения красной сигнальной лампы. Сначала 10 с горит желтый свет (S6), а затем включается и красный свет (S8). Затем, после объединения ветвей, всё начинается с шага S2.

Для выполнения программы создаем приложение (пункт меню Build → Build Solution) и запускаем его в режиме отладки (пункт меню Debug → Start Debugging).

Переходя в окно программы Prog1-POU (рис. 19), мы можем наблюдать процесс ее выполнения в визуальной среде, в которой каждый активный шаг подсвечивается красным цветом в редакторе (своеобразный маркер процесса).

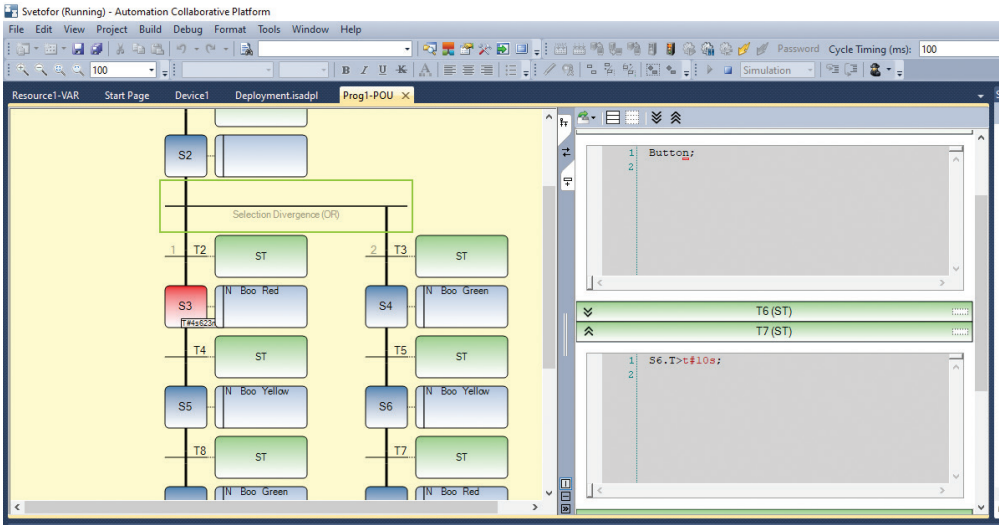

Рис. 19. Демонстрация выполнения программы

В окне переменных Resource1-VAR (рис. 20) можно наблюдать и изменять значения переменных.

В поле Logical Value отображаются значения переменных (например, в данном случае переменная Yellow имеет значение «истина»). Для входных переменных эти значения можно изменять с помощью мышки или клавиатуры. Можно также некоторые переменные отметить в *Spy List* и отслеживать значения только их. Завершение отладки программы осуществляется в пункте меню Debug → Stop Debugging.

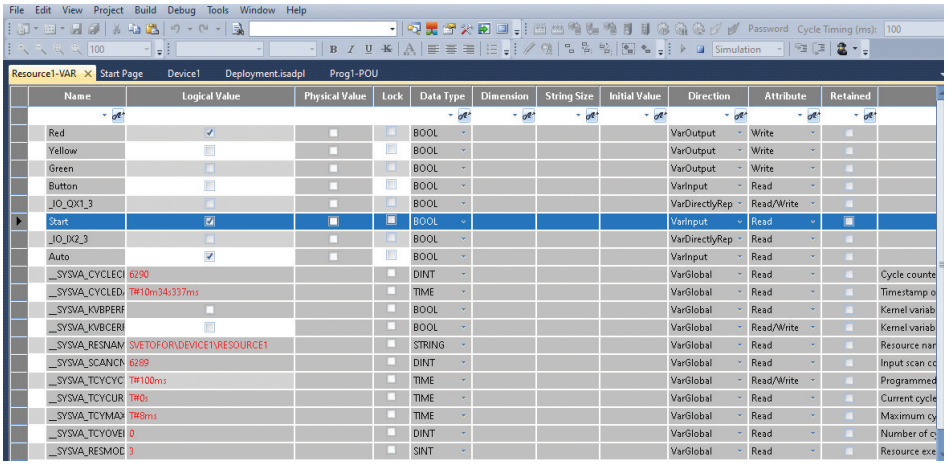

Рис. 20. Окно Resource1-VAR

Практическое задание

Предлагается разработать дорожный контроллер — устройство, обеспечивающее управление светофорами на перекрестке.

Рассмотреть три варианта:

- пешеходный переход с двумя светофорами — для машин и для пешеходов;
- перекресток со светофорами для машин и пешеходов;
- перекресток со светофорами для машин и пешеходов, с дополнительными сигналами бокового движения.

Прототип дорожного контроллера должен быть оборудован портами Ethernet и Wi-Fi для связи с внешней платформой управления. Реализация прототипа дорожного контроллера должна обеспечивать переключение режимов работы светофора в соответствии с заданной программой. Реализация программного интерфейса должна позволять задавать изменение режимов работы дорожного контроллера.

Предусмотреть реализацию адаптивного режима, при котором переключение сигналов светофора зависит от дорожной ситуации, которая определяется с помощью дополнительных сенсоров на перекрестке.

У к а з а н и е : обязательное требование — наличие в программе схождений и расхождений. Таким образом, должно быть предусмотрено либо ветвление (ручной-автоматический, дневной-ночной режим и т. д.), либо параллельная работа отдельных фрагментов программы.

Список библиографических ссылок

1. Голдобин Ю. М., Павлюк Е. Ю. Автоматизация теплоэнергетических установок [Электронный ресурс] : учебное пособие для студентов вуза, обучающихся по направлению подготовки 13.03.01 — Теплоэнергетика и теплотехника. Екатеринбург : УрФУ, 2017. 186 с. URL: http://elar.urfu.ru/handle/10995/55411 (дата обращения: 01.09.2021).

2. Душин И., Тихомиров В. Объектная видеоаналитика реального времени // Открытые системы. СУБД. 2019. № 4. С. 8. URL: https://www.osp.ru/os/2019/04/13055233 (дата обращения: 01.09.2021).

3. Buying NDI. Office of the assistant secretary of defense for production and logistics. Washington, D. C. 20301–8000. Distribution statement A. Approved for public release. Oktober, 1990 SDMP. URL: https://apps.dtic.mil/dtic/tr/fulltext/u2/a232346.pdf (дата обращения: 01.09.2021).

4. Бердичевский М. Е. Конструктивы Евромеханики во встраиваемых системах // Современные технологии автоматизации. 2002. № 4. С. 52–59.

5. Бердичевский М. Е. Универсальные 19″ субблоки серии EuropacPRO // Современные технологии автоматизации. 1998. № 4. С. 64–69.

6. Эрглис К. Э. Интерфейсы открытых систем. М. : Горячая линия-Телеком, 2000. 256 с.

7. Виноградов В. Модульные компактные HPC-системы и сервера ATCA для телекоммуникаций и промышленности. Ч. 1 // Современные технологии автоматизации. 2008. № 4. С. 6–14.

8. Виноградов В. Модульные компактные HPC-системы и сервера ATCA для телекоммуникаций и промышленности. Ч. 2 // Современные технологии автоматизации, 2009. № 1. С. 16–24.

9. Виноградов В. Структура современных встраиваемых модульных систем с сетевой архитектурой // Современные технологии автоматизации. 2008. № 2. С. 6–18.

10. Яковлев В. Базовые принципы построения высокопроизводительных и надежных систем на основе изделий Compact PCI // Современные технологии автоматизации. 2007. № 3. С. 24–30.

11. Борншлегл С. Функциональная безопасность в стандарте Compact PCI 3U // Современные технологии автоматизации. 2015. № 1. С. 12–15.

12. Головастов А. Compact PCI и PXI: не соревнуясь, а дополняя друг друга. Ч. 1 // Современные технологии автоматизации. 2009. № 2. С. 30–35.

13. Головастов А. Compact PCI и PXI: не соревнуясь, а дополняя друг друга. Ч. 2 // Современные технологии автоматизации. 2009. № 3. С. 8–11.

14. Буравлев А. Новые моношоцелевые встраиваемые компьютеры с высокоскоростной шиной VME. СТА // Современные технологии автоматизации. 2008. № 1. С. 52–54.

15. Буравлев А. Спецификация Compact PCI Serial среди открытых спецификаций для построения модульных встраиваемых компьютерных систем // Современные технологии автоматизации. 2012. № 3. С. 84–91.

16. Головастов А. LXI — инструментальный стандарт будущего // Современные технологии автоматизации. 2014. № 4. С.12–21.

17. ГОСТ Р МЭК 61784-3–2015. Промышленные сети. ПРОФИЛИ. Ч. 3. Функциональная безопасность полевых шин. Общие правила и определения профилей. М. : Стандартинформ, 2016.

18. Жданкин В. DART Fieldbus: искробезопасность без ограничений мощности // Современные технологии автоматизации. 2011. № 4. С. 6–10.

19 Решетников И. С., Козлецов А. П., Анисимов Д. Е. ISA-88 — стандарт управления рецептурным производством // Автоматизация в промышленности. 2010. № 4. С. 36–41.

20. Meyer H., Fuchs F., Thiel K. Manufacturing Execution Systems: Optimal Design, Planning, and Deployment. N. Y. : McGraw Hill, 2009. 274 p.

21. Турчанинов В. Программная роботизация в нефтегазовой индустрии // Открытые системы. СУБД. 2019. № 2. С. 24–25.

22. Лопухов И. Сети Real-Time Ethernet: от теории к практической реализации // Современные технологии автоматизации. 2010. № 3. С. 8–15.

23. Сапожников А. От классической полевой шины (fieldbus) EtherCA // Современные технологии автоматизации. 2010. № 3. С. 16–18.

24. Андрюшин А. В., Сабанин В. Р., Смирнов Н. И. Управление и инноватика в теплоэнергетике. М. : МЭИ, 2011. 392 с.

25. ГОСТ 24.104–85. МЕЖГОСУДАРСТВЕННЫЙ СТАНДАРТ. Единая система стандартов автоматизированных систем управления. Автоматизированные системы управления. Общие требования : введ. 1987–01–01. М. : СТАНДАРТИНФОРМ, 2009. 10 с.

26. Интегрированные системы проектирования и управления: SCADA-системы : учебное пособие / И. А. Елизаров [и др.]. Тамбов : Изд-во ФГБОУ ВПО «ТГТУ», 2015. 160 с.

27. ГОСТ Р МЭК 60073–2000. Интерфейс человекомашинный. Маркировка и обозначения органов управления и контрольных устройств. Правила кодирования информации: дата введ. 2002–01–01. М. : Изд-во стандартов, 2001.

28. СТО 59012820.35.240.50.004–2011. Системы диспетчерского управления в электороэнергетике. Система сбора данных и оперативного контроля (SCADA) в диспетчерском управлении : введ. 2011–06–24. М. : ОАО «СО ЕЭС», 2011. 52 с.

29. Дубова Н. Цифровизация с человеческим лицом: «большая семерка» ОС, версия 2020 // Открытые системы. СУБД. 2019. № 4. С. 36.

30. Пандия М. Виртуализация производства: возможности компьютерной симуляции // Control Engineering Россия. 2018. № 3 (75). С. 57–59.

31. Додди С. Основы безопасности индустриальных систем управления // Control Engineering Россия. 2018. № 3 (75). С. 63–65.

32. Лагутенков А. Тихая экспансия интернета вещей // Наука и жизнь. 2018. № 5. С. 38–42.

33. Леонов А. В. Интернет вещей: проблемы безопасности // Омский научный вестник. 2015. № 2 (140). С. 215–218.

34. Варламов И. Г. SCADA нового поколения. Эволюция технологий — революция системостроения // Автоматизация и IT в энергетике. 2016. № 2 (79). С. 28–32.

35. Tryfonas Li, Li H. The Internet of Things: A Security Point of View // Internet Research. 2016. 26 (2). P. 337–359.

36. Scarlett K. Cloud standards: Tools to ensure cloud application interoperability // developerWorks®. IBM Corporation. 2016. 11 с.

37. Разаренов Ф. Облачные технологии в автоматизации: сервис OwenCloud//Автоматизация и производство. 2017. № 2. С. 2–4.

38. Arkian H. R., Diyanat A., Pourkhalili A. MIST: Fog-based data analytics scheme with cost-efficient resource provisioning for IoT crowdsensing applications (англ.) // Journal of Network and Computer Applications. 2017. Vol. 82. P. 152–165.

39. Шеян И. Цифровизация: пусть меня научат // Открытые системы. СУБД. 2018. № 3. С. 39–41.

40. Репин В. Оценка зрелости системы управления бизнес-процессами компании // Открытые системы. СУБД. 2020. № 1. С. 23–25.

41. Середкин И. Интеграция корпоративных систем: RPA // Открытые системы. СУБД. 2019. № 3. С. 30–31. URL: https://www.osp.ru/os/ 2019/03/13055124 (дата обращения: 01.09.2021).

42. Загидуллин Р. Р. Управление машиностроительным производством с помощью систем MES, APS, ERP. Старый Оскол : ТНТ, 2011. 372 с.

43. Решетников И. С., Козлецов А. П., Анисимов Д. Е. ISA-88 — стандарт управления рецептурным производством // Автоматизация в промышленности. 2010. № 4. С. 36–41.

44. Турчанинов В. Программная роботизация в нефтегазовой индустрии [Электронный ресурс] // Открытые системы. СУБД. 2019. № 2. С. 24–25. URL: https://www.osp.ru/os/2019/02/13054963 (дата обращения: 01. 09.2021).

45. Бейдер А. RPA: что, где, кому? [Электронный ресурс] // Открытые системы. СУБД. 2020. № 1. С. 42–43. URL: https://www.osp.ru/os/2020/01/13055336 (дата обращения: 01. 09.2021).

46. Цилюрик О., Горошко Е. QNX/UNIX: Анатомия параллелизма. СПб. : Символ-Плюс, 2006. 288 с.

47. Стивенс У. UNIX: взаимодействие процессов : пер. с англ. СПб. : Питер, 2002. 576 с.

48. Зыль С. Н. Операционная система реального времени QNX Neutrino 6.5.0. Системная архитектура : пер. с англ. СПб. : БХВ-Петербург, 2014. 400 с.

49. Кертен Р. Введение в QNX/Neutrino 2: Руководство по программированию приложений реального времени в QNX Realtime Platform : пер. с англ. СПб. : Петрополис, 2001. 479 с.

50. Федосеев А. Системы реального времени: от Linux к Java // Открытые системы. СУБД. 2009. № 3. С. 16–24.

Приложение (справочное)

Список зарубежных организаций, комитетов и комиссий, формирующих стандарты на технологии, компоненты и комплексы ИУС

ISO (МОС) — Международная организация по стандартам, независимая неправительственная организация, в которую входят представители 163 стран. Действует около 70 лет. Это крупнейший в мире разработчик добровольных международных технологических стандартов.

IEC (МЭК) — Международная электротехническая комиссия, действующая более 100 лет. Комиссия является главной движущей силой разработки международных стандартов по всем технологиям с использованием электрических, электронных и смежных процессов.

IEEE — международная некоммерческая ассоциация специалистов в области техники. Является мировым лидером в области разработки стандартов по радиоэлектронике, электротехнике и аппаратному обеспечению вычислительных систем и сетей.

ANSI (American national standards institute) — объединение американских промышленных и деловых групп, разрабатывающее торговые и коммуникационные стандарты. Входит в организации ISO и IEC, где представляет интересы США.

EIA (Альянс отраслей электронной промышленности) — профессиональная организация, расположенная в США и разрабатывающая электрические и функциональные стандарты с идентификатором RS (Recommended Standards).

Сертификат **CE** (Conformité Européenne) — это документ, служащий подтверждением соответствия продукции требованиям Европейского союза.

VDE (Verband der Elektrotechnik) — институт или объединение немецких инженеров по электронике, электротехнике и информатике.

DIN (Deutsches Institut für Normung) — Немецкий институт по стандартизации. Членами DIN являются различные предприятия, союзы, государственные организации, торговые фирмы и научные институты, которые накопили значительный опыт в разработках нормативных документов.

BSI (British Standards Institution) — британская организация, занимающаяся координацией деятельности по разработке стандартов на основе соглашения между всеми заинтересованными сторонами и принятием стандартов.

NEMA — американская национальная ассоциация производителей электронного оборудования.

CSA — канадская ассоциация по стандартам.

UL — сертификационные лаборатории.

CERN — Европейский институт ядерных исследований.

CSCC (Cloud Standards Customer Council) — информационно-пропагандистская группа конечных пользователей, которая стремится «ускорить успешное внедрение облака» как средства укрепления предприятий XXI в.

DMTF (Distributed Management Task Force) — ассоциация IT-компаний и специалистов, сотрудничающих в области разработки и продвижения стандартов администрирования и взаимодействия корпоративных систем.

Open Cloud Standards Incubator — рабочая группа, разрабатывающая стандарты на облачные сервисы по организационному взаимодействию между внутрикорпоративными частными, общедоступными и гибридными облаками.

CMWG (Cloud Management Working Group) — рабочая группа, занимающаяся визуальным представлением всего жизненного цикла облачной службы.

CADF (Cloud Auditing Data Federation Working Group) — рабочая группа по стандартизации «событий аудита для всех поставщиков облаков и услуг» в целях решения важных проблем в сфере облачных вычислений, вызванных несоответствием или несовместимостью.

ETSI (European Telecommunications Standards Institute) — организация, которая создает международные стандарты в сфере информационных и коммуникационных технологий для улучшения взаимодействия систем, повышения их эффективности и экономии за счет обмена знаниями и опытом.

CSC (Cloud Standards Coordination) — организация-координатор по облачным стандартам в области разработки средств взаимодействия приложений в облаке.

GICTF (Global Inter-Cloud Technology Forum) — организация по содействию стандартизации сетевых протоколов и интерфейсов для создания более надежной сети облачных услуг, которая решает проблемы безопасности, качества данных, системы реагирования и надежности.

ITU — Международный союз электросвязи, являющийся специализированным учреждением Организации Объединенных Наций. Занимается разработкой технических стандартов, обеспечивающих сетевое взаимодействие.

JCA-Cloud (Joint Coordination Activity on Cloud Computing) — подгруппа, координирующая работу по стандартизации облачных вычислений в рамках ITU и других организаций.

NIST — Национальный институт стандартов и технологии США. Относится к Министерству торговли США, работает над развитием метрологии, стандартов и технологий. NIST определяет технологии облачных вычислений для государственных учреждений и промышленности.

OGF (Open Grid Forum) — международная группа ИТ-специалистов, работающих методом открытых форумов и мероприятий в сфере быстрой разработки и развертывания передовых прикладных распределенных вычислительных сред, таких как облако, grid-сети и смежные средства хранения данных и сетевой связи.

OMG (Object Management Group) — Международный консорциум по технологическим стандартам, который первоначально был нацелен на стандартизацию распределенных объектно ориентированных систем, на программы моделирования, системы и бизнес-процессы и на создание стандартов, основанных на моделях.

OCC (Open Cloud Consortium) — организация университетов, компаний и государственных институтов, которая поддерживает исследования в области медицины, здравоохранения, естественных наук и экологии, администрируя и эксплуатируя облачную вычислительную инфраструктуру.

OASIS (Организация по совершенствованию стандартов структурированной информации) — консорциум, который объединяет членов более чем из 65 стран и продвигает облачные протоколы и стандарты.

SNIA (Storage Networking Industry Association) — это международная группа, специализирующаяся на разработке стандартов и технологий для управления информацией и ее хранения.

ARTS (Association for Retail Technology Standards) — подразделение Национальной федерации розничной торговли США, которая стремится уменьшить стоимость ИТ за счет внедрения стандартов.

TM Forum — международная торговая ассоциация, которая продвигает идею ИТ как службы через свою группу Cloud Forum. Эта группа проводит исследования, испытания и планирование отраслевых мероприятий и предлагает литературу по передовому опыту, а также интерфейсы на основе стандартов программного обеспечения, учебные курсы, труды конференций и публикации.

Учебное издание

Александрова Ольга Николаевна
Ваулин Сергей Степанович
Папуловская Наталья Владимировна

ИНФОРМАЦИОННО-УПРАВЛЯЮЩИЕ СИСТЕМЫ: АРХИТЕКТУРА И РАЗРАБОТКА

Редактор И. В. Меркурьева
Верстка Е.В. Ровнушкиной

Подписано в печать 27.10.2021. Формат 70×100/16.
Бумага писчая. Цифровая печать. Усл. печ. л. 12,0.
Уч.-изд. л. 9,4. Тираж 30 экз. Заказ 240.

Издательство Уральского университета
Редакционно-издательский отдел ИПЦ УрФУ
620049, Екатеринбург, ул. С. Ковалевской, 5
Тел. 8 (343) 375-48-25, 375-46-85, 374-19-41
E-mail rio@urfu.ru

Отпечатано в Издательско-полиграфическом центре УрФУ
620075, Екатеринбург, ул. Тургенева, 4
Тел.: 8 (343) 350-56-64, 350-90-13
Факс: 8 (343) 358-93-06
E-mail: press-urfu@mail.ru

9 785799 633387

АЛЕКСАНДРОВА ОЛЬГА НИКОЛАЕВНА
Кандидат физико-математических наук, Doctor rerum naturalium, доцент центра ускоренного обучения ИРИТ-РТФ УрФУ.

ВАУЛИН СЕРГЕЙ СТЕПАНОВИЧ
Кандидат технических наук, доцент центра ускоренного обучения ИРИТ-РТФ УрФУ.

ПАПУЛОВСКАЯ НАТАЛЬЯ ВЛАДИМИРОВНА
Кандидат педагогических наук, доцент кафедры информационных технологий и систем управления ИРИТ-РТФ УрФУ.